

| 学习目标

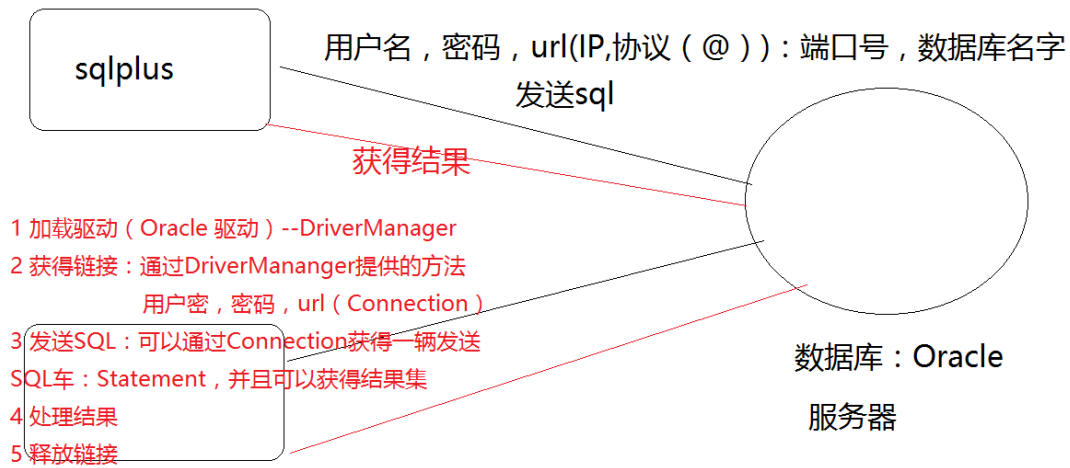
- * 能够理解JDBC的概述
- * 能够掌握JDBC常见的API
- * 能够掌握JDBC的HelloWorld的开发
- * 能够掌握JDBC工具类的编写
- * 能够掌握JDBC的CRUD的编写
- * 能够使用PreparedStatement防止SQL注入
- * 能够掌握JDBC的批处理

* 回顾

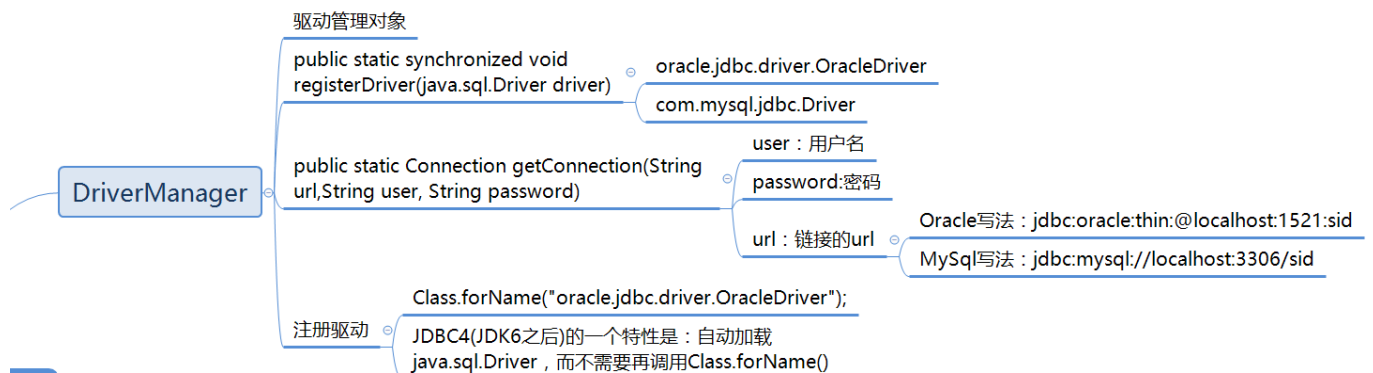
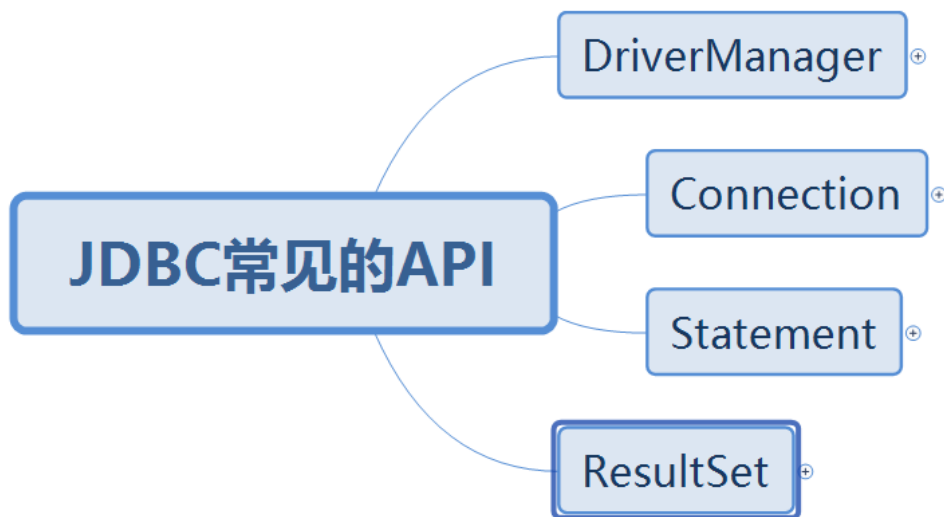
- * DDL : create table,alter table modify,add,drop,rename... to...
drop table,truncate table, delete
- * DML: insert,update,delete
- * DQL: select * from table_name where condition group by,having,order by(asc desc),limit
- * JDBC : JAVA DataBase Connectivity
- * 常见的API : DriverManager , Connection , Statement , ResultSet
- * HelloWorld JDBC

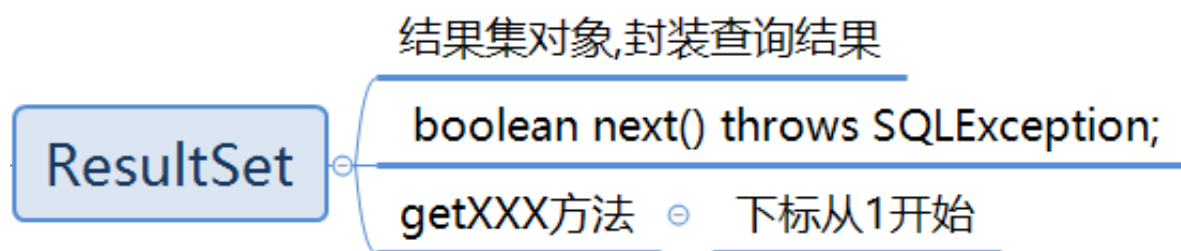
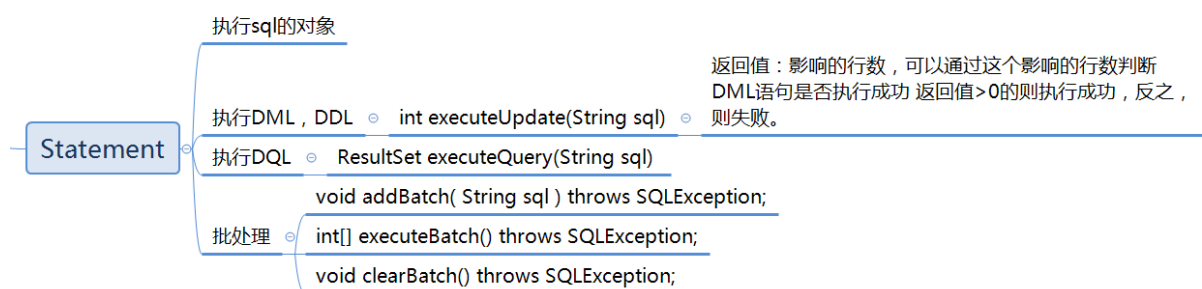
* 能够理解JDBC的概述

- * JDBC : Java DataBase Connectivity (Java数据库连接)
- * 由sun公司提供一套Java操作数据库的接口标准，而且由各大厂商提供驱动(jar 包)。
 - * Oracle提供Oracle驱动的实现 (jar包)
 - * MySQL提供MySQL驱动的实现 (jar包)
- * 能够掌握JDBC常见的API



Java程序





* 能够掌握JDBC的HelloWorld的开发

```

1 public static void main(String[] args) throws SQLException {
2     // 1 加载驱动 (Oracle 驱动) --DriverManager
3     DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
4     // 2 获得链接: 通过DriverMananger提供的方法
5     //     用户密, 密码, url (Connection)
6     String user="scott";
7     String password="tiger";
8     String url="jdbc:oracle:thin:@192.168.1.121:1521:orcl";
9     Connection conn = DriverManager.getConnection(url, user, password);

```

```

10 //      3 发送SQL: 可以通过Connection获得一辆发送SQL车: Statement, 并且可以获得结果
11      Statement statement = conn.createStatement();
12      String sql="select * from emp";
13      ResultSet rs=statement.executeQuery(sql);
14 //      4 处理结果
15      while(rs.next()) {
16          String empNo=rs.getString(1);// 数组下标是从开始, JDBC不从零开始,从1开
17          String eName=rs.getString(2);
18          String job=rs.getString(3);
19          String mgr=rs.getString(4);
20          String hireDate=rs.getString(5);
21          String sal=rs.getString(6);
22          String comm=rs.getString(7);
23          String deptNo=rs.getString(8);
24          System.out.println(empNo+":"+eName+":"+job+":"+mgr+":"+hireDate+":'
25      }
26 //      5 释放链接
27      rs.close();
28      conn.close();
29  }
30 结果:
31
32 7369:SMITH:CLERK:7902:1980-12-17 00:00:00:800:null:20
33 7499:ALLEN:SALESMAN:7698:1981-02-20 00:00:00:1600:300:30
34 7521:WARD:SALESMAN:7698:1981-02-22 00:00:00:1250:500:30
35 7566:JONES:MANAGER:7839:1981-04-02 00:00:00:2975:null:20
36 7654:MARTIN:SALESMAN:7698:1981-09-28 00:00:00:1250:1400:30
37 7698:BLAKE:MANAGER:7839:1981-05-01 00:00:00:2850:null:30
38 7782:CLARK:MANAGER:7839:1981-06-09 00:00:00:2450:null:10
39 7788:SCOTT:ANALYST:7566:1987-04-19 00:00:00:3000:null:20
40 7839:KING:PRESIDENT:null:1981-11-17 00:00:00:5000:null:10
41 7844:TURNER:SALESMAN:7698:1981-09-08 00:00:00:1500:0:30
42 7876:ADAMS:CLERK:7788:1987-05-23 00:00:00:1100:null:20
43 7900:JAMES:CLERK:7698:1981-12-03 00:00:00:950:null:30
44 7902:FORD:ANALYST:7566:1981-12-03 00:00:00:3000:null:20
45 7934:MILLER:CLERK:7782:1982-01-23 00:00:00:1300:null:10
46
47 * 注册的方式
48 * Class.forName("oracle.jdbc.driver.OracleDriver");
49 * jdk1.6之后: 注册驱动, 可以省略不写

```

```

50
51 * 正确关闭释放资源的方法
52 public static void main(String[] args){
53     String user="scott";
54     String password="tiger";
55     String url="jdbc:oracle:thin:@192.168.1.121:1521:orcl";
56     Connection conn = null;
57     Statement statement =null;
58     ResultSet rs=null;
59     try {
60         conn = DriverManager.getConnection(url, user, password);
61 //      3 发送SQL: 可以通过Connection获得一辆发送SQL车: Statement, 并且可以获得
62         statement = conn.createStatement();
63         String sql="select * from emp";
64         rs=statement.executeQuery(sql);
65 //      4 处理结果
66         while(rs.next()) {
67             String empNo=rs.getString(1);// 数组下标是从开始, JDBC不从零开始,
68             String eName=rs.getString(2);
69             String job=rs.getString(3);
70             String mgr=rs.getString(4);
71             String hireDate=rs.getString(5);
72             String sal=rs.getString(6);
73             String comm=rs.getString(7);
74             String deptNo=rs.getString(8);
75             System.out.println(empNo+":"+eName+":"+job+":"+mgr+":"+hireDate);
76         }
77     } catch (SQLException e) {
78         e.printStackTrace();
79     }finally {
80 //      5 释放链接
81         if(rs!=null) {
82             try {
83                 rs.close();
84             } catch (SQLException e) {
85                 e.printStackTrace();
86             }
87         }
88         if(statement!=null) {
89             try {

```

```

90         statement.close();
91     } catch (SQLException e) {
92         e.printStackTrace();
93     }
94 }
95 if(conn!=null) {
96     try {
97         conn.close();
98     } catch (SQLException e) {
99         e.printStackTrace();
100    }
101 }
102
103 }
104 }
105

```

* 能够掌握JDBC工具类的编写

```

1 public class ConnectionUtils {
2     private static String userName;
3     private static String password;
4     private static String url;
5
6     static {
7         Properties prop=new Properties();
8         try {
9             prop.load(ConnectionUtils.class.getClassLoader().getResourceAsStream("config.properties"));
10            userName=prop.getProperty("username");
11            password=prop.getProperty("password");
12            url=prop.getProperty("url");
13        } catch (IOException e) {
14            e.printStackTrace();
15        }
16    }
17
18    public static Connection getConnection() {

```

```
19     try {
20         return DriverManager.getConnection(url, userName, password);
21     } catch (SQLException e) {
22         e.printStackTrace();
23     }
24     return null;
25 }
26
27
28 /**
29  * 是否资源的方法
30  * @param conn
31  * @param st
32  * @param rs
33  */
34 public static void close(Connection conn,Statement st,ResultSet rs) {
35     if(rs!=null) {
36         try {
37             rs.close();
38         } catch (SQLException e) {
39             e.printStackTrace();
40         }
41     }
42     if(st!=null) {
43         try {
44             st.close();
45         } catch (SQLException e) {
46             e.printStackTrace();
47         }
48     }
49     if(conn!=null) {
50         try {
51             conn.close();
52         } catch (SQLException e) {
53             e.printStackTrace();
54         }
55     }
56
57 }
58
```

```

59     public static void closeRs(ResultSet rs) {
60         close(null, null, rs);
61     }
62
63     public static void closeSt(Statement st) {
64         close(null, st, null);
65     }
66
67     public static void closeConn(Connection con) {
68         close(con, null, null);
69     }
70 }
71
72 * 测试
73 public static void main(String[] args){
74     Connection conn = null;
75     Statement st =null;
76     ResultSet rs=null;
77     try {
78         conn=ConnectionUtils.getConnection();
79 //        3 发送SQL: 可以通过Connection获得一辆发送SQL车: Statement, 并且可以获得
80         st = conn.createStatement();
81         String sql="select * from emp";
82         rs=st.executeQuery(sql);
83 //        4 处理结果
84         while(rs.next()) {
85             String empNo=rs.getString(1);// 数组下标是从开始, JDBC不从零开始,
86             String eName=rs.getString(2);
87             String job=rs.getString(3);
88             String mgr=rs.getString(4);
89             String hireDate=rs.getString(5);
90             String sal=rs.getString(6);
91             String comm=rs.getString(7);
92             String deptNo=rs.getString(8);
93             System.out.println(empNo+":"+eName+":"+job+":"+mgr+":"+hireDate);
94         }
95     } catch (SQLException e) {
96         e.printStackTrace();
97     }finally {
98         ConnectionUtils.close(conn, st, rs);

```



```
99     }
100 }
101
```

* 能够掌握JDBC的CRUD的编写

```
1 package com.lg.test1;
2
3 import java.sql.Connection;
4 import java.sql.ResultSet;
5 import java.sql.SQLException;
6 import java.sql.Statement;
7
8 import org.junit.After;
9 import org.junit.Before;
10 import org.junit.Test;
11
12 public class Test2 {
13
14     private Connection conn;
15     private Statement st;
16     private ResultSet rs;
17     @Before
18     public void setUp() throws SQLException {
19         conn = ConnectionUtils.getConnection();
20         st = conn.createStatement();
21     }
22
23     @Test
24     public void testAdd() throws SQLException {
25         String sql = "INSERT INTO stu(sid,sname,age,gender) VALUES('S_1013','xi
26         // 3 insert
27         int result = st.executeUpdate(sql);
28         if (result > 0) {
29             System.out.println("insert ok");
30         }
31     }
```

```
32
33 @Test
34 public void testUpdate() throws SQLException {
35     String sql = "UPDATE stu SET sname='xiaohong' WHERE sid='S_1013'";
36     // 3 insert
37     int result = st.executeUpdate(sql);
38     if (result > 0) {
39         System.out.println("UPDATE ok");
40     }
41 }
42
43 @Test
44 public void testDelete() throws SQLException {
45     String sql = "DELETE FROM stu WHERE sid='S_1013'";
46     // 3 insert
47     int result = st.executeUpdate(sql);
48     if (result > 0) {
49         System.out.println("DELETE ok");
50     }
51 }
52
53 @Test
54 public void testQueryAll() throws SQLException {
55     String sql = "SELECT * FROM stu;
56     // 3 insert
57     ResultSet rs = st.executeQuery(sql);
58     // st.executeUpdate(sql)
59     // 5 处理结果
60     while (rs.next()) {
61         // JDBC 下标是从1开始的
62         String sid = rs.getString(1);
63         String sname = rs.getString(2);
64         int age = rs.getInt(3);
65         String sex = rs.getString(4);
66         System.out.println(sid + ":" + sname + ":" + age + ":" + sex);
67     }
68 }
69
70 @After
71 public void finish() {
```

```
72         ConnectionUtils.close(conn, st, rs);
73     }
74 }
75
76 * StudentDao的编写
77 * DAO(Data Access Object) 数据访问对象是一个面向对象的数据库接口
78 public class Student {
79     private String sid;
80     private String name;
81     private int age;
82     private String gender;
83     public Student() {
84         super();
85     }
86
87     public Student(String sid, String name, int age, String gender) {
88         super();
89         this.sid = sid;
90         this.name = name;
91         this.age = age;
92         this.gender = gender;
93     }
94
95     public String getSid() {
96         return sid;
97     }
98     public void setSid(String sid) {
99         this.sid = sid;
100    }
101    public String getName() {
102        return name;
103    }
104    public void setName(String name) {
105        this.name = name;
106    }
107    public int getAge() {
108        return age;
109    }
110    public void setAge(int age) {
111        this.age = age;
```

```
112     }
113     public String getGender() {
114         return gender;
115     }
116     public void setGender(String gender) {
117         this.gender = gender;
118     }
119
120     @Override
121     public int hashCode() {
122         final int prime = 31;
123         int result = 1;
124         result = prime * result + age;
125         result = prime * result + ((gender == null) ? 0 : gender.hashCode());
126         result = prime * result + ((name == null) ? 0 : name.hashCode());
127         result = prime * result + ((sid == null) ? 0 : sid.hashCode());
128         return result;
129     }
130
131     @Override
132     public boolean equals(Object obj) {
133         if (this == obj)
134             return true;
135         if (obj == null)
136             return false;
137         if (getClass() != obj.getClass())
138             return false;
139         Student other = (Student) obj;
140         if (age != other.age)
141             return false;
142         if (gender == null) {
143             if (other.gender != null)
144                 return false;
145         } else if (!gender.equals(other.gender))
146             return false;
147         if (name == null) {
148             if (other.name != null)
149                 return false;
150         } else if (!name.equals(other.name))
151             return false;
```

```
152         if (sid == null) {
153             if (other.sid != null)
154                 return false;
155         } else if (!sid.equals(other.sid))
156             return false;
157         return true;
158     }
159
160     @Override
161     public String toString() {
162         return "User [sid=" + sid + ", name=" + name + ", age=" + age + ", genc
163     }
164
165 }
166
167 public interface StudentDao {
168     // CRUD
169     // 添加学生
170     boolean add(Student student);
171
172     // 根据学号，删除学生
173     boolean delStudent(String sid);
174
175     // 通过学号，修改姓名
176     boolean updateStudent(String sid,String name);
177
178     // 通过学号获得的学生
179     public Student getStudent(String sid);
180
181     // 查询所有的学生
182     public List<Student> getStudents();
183
184     // 查询部分学生(分页)
185     public List<Student> getStudents(int offset,int row);
186 }
187
188 public class StudentDaoImpl implements StudentDao {
189
190     @Override
191     public boolean add(Student student) {
```

```

192     String sql = "INSERT INTO stu(sid,sname,age,gender) VALUES('" + student
193         + "',''" + student.getAge() + "',''" + student.getGender() + "')"
194     Connection conn = ConnectionUtils.getConnection();
195     Statement st = null;
196     int result = 0;
197     try {
198         st = conn.createStatement();
199         result = st.executeUpdate(sql);
200     } catch (SQLException e) {
201         e.printStackTrace();
202     } finally {
203         ConnectionUtils.close(conn, st, null);
204     }
205     return result > 0;
206 }
207
208 @Override
209 public boolean delStudent(String sid) {
210     Connection conn = ConnectionUtils.getConnection();
211     Statement st = null;
212     String sql = "DELETE FROM stu WHERE sid='" + sid + "'";
213     // 3 insert
214     int result = 0;
215     try {
216         st = conn.createStatement();
217         result = st.executeUpdate(sql);
218     } catch (SQLException e) {
219         e.printStackTrace();
220     } finally {
221         ConnectionUtils.close(conn, st, null);
222     }
223     return result > 0;
224 }
225
226 @Override
227 public boolean updateStudent(String sid, String name) {
228     Connection conn = ConnectionUtils.getConnection();
229     String sql = "UPDATE stu SET sname='" + name + "' WHERE sid='" + sid + "'";
230     Statement st = null;
231     int result = 0;

```

```

232     try {
233         st = conn.createStatement();
234         result = st.executeUpdate(sql);
235     } catch (SQLException e) {
236         e.printStackTrace();
237     } finally {
238         ConnectionUtils.close(conn, st, null);
239     }
240     return result > 0;
241 }
242
243 @Override
244 public Student getStudent(String sid) {
245     Connection conn = ConnectionUtils.getConnection();
246     String sql = "select * from stu where sid='" + sid + "'";
247     Statement st = null;
248     ResultSet rs = null;
249     Student student = null;
250     try {
251         st = conn.createStatement();
252         rs = st.executeQuery(sql);
253         if (rs.next()) {
254             student = new Student(rs.getString(1), rs.getString(2), Integer
255                 rs.getString(4));
256         }
257     } catch (SQLException e) {
258         e.printStackTrace();
259     } finally {
260         ConnectionUtils.close(conn, st, rs);
261     }
262     return student;
263 }
264
265 @Override
266 public List<Student> getStudents() {
267     Connection conn = ConnectionUtils.getConnection();
268     String sql = "select * from stu";
269     Statement st = null;
270     ResultSet rs = null;
271     List<Student> students = new ArrayList<Student>();

```

```

272     try {
273         st = conn.createStatement();
274         rs = st.executeQuery(sql);
275         while (rs.next()) {
276             try {
277                 Student student = new Student(rs.getString(1), rs.getString(2),
278                     rs.getString(3), rs.getString(4));
279                 students.add(student);
280             } catch (NumberFormatException e) {
281                 // e.printStackTrace();
282                 Student student=new Student();
283                 student.setSid(rs.getString(1));
284                 student.setName(rs.getString(2));
285                 students.add(student);
286             }
287         }
288     } catch (SQLException e) {
289         e.printStackTrace();
290     } finally {
291         ConnectionUtils.close(conn, st, rs);
292     }
293     return students;
294 }
295
296 @Override
297 public List<Student> getStudents(int offset, int row) {
298     Connection conn = ConnectionUtils.getConnection();
299     String sql = "select sid,sname,age,gender from (select rownum rn,sid,sname,age,gender from emp)";
300     Statement st = null;
301     ResultSet rs = null;
302     List<Student> students = new ArrayList<Student>();
303     try {
304         st = conn.createStatement();
305         rs = st.executeQuery(sql);
306         while (rs.next()) {
307             try {
308                 Student student = new Student(rs.getString(1), rs.getString(2),
309                     rs.getString(3), rs.getString(4));
310                 students.add(student);
311             } catch (NumberFormatException e) {

```



```
312 //          e.printStackTrace();
313          Student student=new Student();
314          student.setSid(rs.getString(1));
315          student.setName(rs.getString(2));
316          students.add(student);
317      }
318  }
319  } catch (SQLException e) {
320      e.printStackTrace();
321  } finally {
322      ConnectionUtils.close(conn, st, rs);
323  }
324  return students;
325  }
326
327 }
328
329 public class StudentDaoTest {
330     StudentDao studentDao;
331     @Before
332     public void setUp() {
333         studentDao=new StudentDaoImpl();
334     }
335     @Test
336     public void testAdd() {
337         studentDao.add(new Student("S_1012", "xiaoming", 20, "male"));
338     }
339     @Test
340     public void testUpdate() {
341         studentDao.updateStudent("S_1012", "xiaobai");
342     }
343     @Test
344     public void testDel() {
345         studentDao.delStudent("S_1012");
346     }
347
348     @Test
349     public void testGetStudent() {
350         Student student = studentDao.getStudent("S_1009");
351         System.out.println(student);
```

```

352     }
353     @Test
354     public void testGetStudents() {
355         List<Student> students = studentDao.getStudents();
356         for(Student student:students) {
357             System.out.println(student);
358         }
359     }
360     @Test
361     public void testGetStudents2() {
362         List<Student> students = studentDao.getStudents(2,5);
363         for(Student student:students) {
364             System.out.println(student);
365         }
366     }
367 }
368

```

* 能够使用PreparedStatement防止SQL注入

```

1  * 创建用户表
2  create table t_user(
3      username varchar2(50) primary key not null,
4      password varchar2(50)
5  );
6  insert into t_user values('xiaohei','202cb962ac59075b964b07152d234b70');
7  insert into t_user values('xiaobai','202cb962ac59075b964b07152d234b70');
8  commit;
9
10 public class User {
11     private String name;
12     private String password;
13     public User() {
14         super();
15     }
16
17     public User(String name, String password) {
18         super();
19         this.name = name;

```

```
20     this.password = password;
21 }
22
23 public String getName() {
24     return name;
25 }
26 public void setName(String name) {
27     this.name = name;
28 }
29 public String getPassword() {
30     return password;
31 }
32 public void setPassword(String password) {
33     this.password = password;
34 }
35 @Override
36 public int hashCode() {
37     final int prime = 31;
38     int result = 1;
39     result = prime * result + ((name == null) ? 0 : name.hashCode());
40     result = prime * result + ((password == null) ? 0 : password.hashCode());
41     return result;
42 }
43
44 @Override
45 public boolean equals(Object obj) {
46     if (this == obj)
47         return true;
48     if (obj == null)
49         return false;
50     if (getClass() != obj.getClass())
51         return false;
52     User other = (User) obj;
53     if (name == null) {
54         if (other.name != null)
55             return false;
56     } else if (!name.equals(other.name))
57         return false;
58     if (password == null) {
59         if (other.password != null)
```

```

60         return false;
61     } else if (!password.equals(other.password))
62         return false;
63     return true;
64 }
65
66 @Override
67 public String toString() {
68     return "User [name=" + name + ", password=" + password + "]";
69 }
70 }
71
72 public interface UserDao {
73     // 可以SQL注入的
74     User getUser(String username,String password);
75     // 防止SQL注入的
76     User getUser2(String username,String password);
77 }
78
79 public class UserDaoImpl implements UserDao{
80
81     @Override
82     public User getUser(String username, String password) {
83         Connection conn = ConnectionUtils.getConnection();
84         String sql = "select * from t_user where username='" + username + "' and pa
85         Statement st = null;
86         ResultSet rs = null;
87         User user = null;
88         System.out.println(sql);
89         try {
90             st = conn.createStatement();
91             rs = st.executeQuery(sql);
92             if (rs.next()) {
93                 user = new User(rs.getString(1), rs.getString(2));
94             }
95         } catch (SQLException e) {
96             e.printStackTrace();
97         } finally {
98             ConnectionUtils.close(conn, st, rs);
99         }

```

```
100     return user;
101 }
102
103
104 @Override
105 public User getUser2(String username, String password) {
106     Connection conn = ConnectionUtils.getConnection();
107     String sql = "select * from t_user where username=? and password=?";
108     PreparedStatement st = null;
109     ResultSet rs = null;
110     User user = null;
111     System.out.println(sql);
112     try {
113         // 先预编译好
114         st = conn.prepareStatement(sql);
115         st.setString(1, username);
116         st.setString(2, password);
117         rs = st.executeQuery();
118         if (rs.next()) {
119             user = new User(rs.getString(1), rs.getString(2));
120         }
121     } catch (SQLException e) {
122         e.printStackTrace();
123     } finally {
124         ConnectionUtils.close(conn, st, rs);
125     }
126     return user;
127 }
128
129 }
130
131 * 测试
132 public class MD5Utils {
133     /**
134      * 使用md5的算法进行加密
135      */
136     public static String md5(String plainText) {
137         byte[] secretBytes = null;
138         try {
139             secretBytes = MessageDigest.getInstance("md5").digest(
```

```

140         plainText.getBytes());
141     } catch (NoSuchAlgorithmException e) {
142         throw new RuntimeException("没有md5这个算法!");
143     }
144     String md5code = new BigInteger(1, secretBytes).toString(16); // 16进制数
145     // 如果生成数字未满32位，需要前面补0
146     for (int i = 0; i < 32 - md5code.length(); i++) {
147         md5code = "0" + md5code;
148     }
149     return md5code;
150 }
151
152 }
153
154 @Test
155 public void test1() {
156     UserDao userDao = new UserDaoImpl();
157     // User user = userDao.getUser("xiaohei", MD5Utils.md5("123"));
158     // User user = userDao.getUser("xiaohei123", MD5Utils.md5("56")+"'" or '1'="
159     // select * from t_user where username='xiaohei123' and password='9f61408c
160     // User user = userDao.getUser2("xiaohei", MD5Utils.md5("123"));
161     User user = userDao.getUser2("xiaohei123", MD5Utils.md5("56")+"'" or '1'="
162     if (user != null) {
163         System.out.println("登录成功");
164     } else {
165         System.out.println("登录失败");
166     }
167 }
168
169 * sql注入
170 select * from t_user where username='xiaohei123' and password='202cb962ac59075t

```

* 能够掌握JDBC的批处理

* 概述：程序向数据库发送一批SQL语句执行，这时应避免向数据库一条条的发送执行，提升执行效率。

* JDBC实现批处理有两种方式：Statement和PreparedStatement

* Statement.addBatch(sql)方式实现批处理：

* 优点：可以向数据库发送多条不同的SQL语句。

* 缺点：

* SQL语句没有预编译。

* 当向数据库发送多条语句相同，但仅参数不同的SQL语句时，需重复写上很多条SQL语句。

* 采用PreparedStatement.addBatch()实现批处理

* 优点：发送的是预编译后的SQL语句，执行效率高。

* 缺点：只能应用在SQL语句相同，但参数不同的批处理中。因此此种形式的批处理经常用于在同一个表中批量插入数据，或批量更新表的数据。

```
1 * 创建表testbatch表
2 create table testbatch(
3     name varchar2(50)
4 );
5 * Statement 批处理
6 @Test
7 public void test1() throws SQLException {
8     // 1 获得连接
9     Connection conn = ConnectionUtils.getConnection();
10    String sql1 = "insert into testbatch(name) values('xiaohei0')";
11    String sql2 = "insert into testbatch(name) values('xiaohei1')";
12    String sql3 = "insert into testbatch(name) values('xiaohei2')";
13    String sql4 = "insert into testbatch(name) values('xiaohei3')";
14    String sql5 = "insert into testbatch(name) values('xiaohei4')";
15    String sql6 = "insert into testbatch(name) values('xiaohei5')";
16    Statement st = conn.createStatement();
17    st.addBatch(sql1);
18    st.addBatch(sql2);
19    st.addBatch(sql3);
20    st.addBatch(sql4);
21    st.addBatch(sql5);
22    st.addBatch(sql6);
23    // 打包一起发过去
24    st.executeBatch();
25    st.clearBatch();
```

```
26         ConnectionUtils.close(conn, st, null);
27     }
28
29 * 使用PreparedStatement的批处理
30 * 不使用批处理花的时间:
31     @Test
32     public void test2() throws SQLException {
33         long start = System.currentTimeMillis();
34         // 1 获得连接
35         Connection conn = ConnectionUtils.getConnection();
36         String sql = "insert into testbatch(name) values(?)";
37         PreparedStatement psmt = conn.prepareStatement(sql);
38         for (int i = 1; i < 10008; i++) {
39             psmt.setString(1, "xiaobai" + i);
40             psmt.executeUpdate();
41         }
42         long end = System.currentTimeMillis();
43         System.out.println("不使用批处理花的时间:" + (end - start));
44         ConnectionUtils.close(conn, psmt, null);
45     }
46 * 结果:
47     不使用批处理花的时间:11935
48 * 使用批处理
49     @Test
50     public void test2() throws SQLException {
51         long start = System.currentTimeMillis();
52         // 1 获得连接
53         Connection conn = ConnectionUtils.getConnection();
54         String sql = "insert into testbatch(name) values(?)";
55         PreparedStatement psmt = conn.prepareStatement(sql);
56         for (int i = 1; i < 10008; i++) {
57             psmt.setString(1, "xiaobai" + i);
58             psmt.addBatch();
59             // 每次发一千条
60             if (i % 1000 == 0) {
61                 psmt.executeBatch();
62                 psmt.clearBatch();
63             }
64         }
65         psmt.executeBatch();
```



```
66         psmt.clearBatch();
67         long end = System.currentTimeMillis();
68         System.out.println("使用批处理花的时间:" + (end - start));
69         ConnectionUtils.close(conn, psmt, null);
70     }
71 * 结果:
72     使用批处理花的时间:848
```