

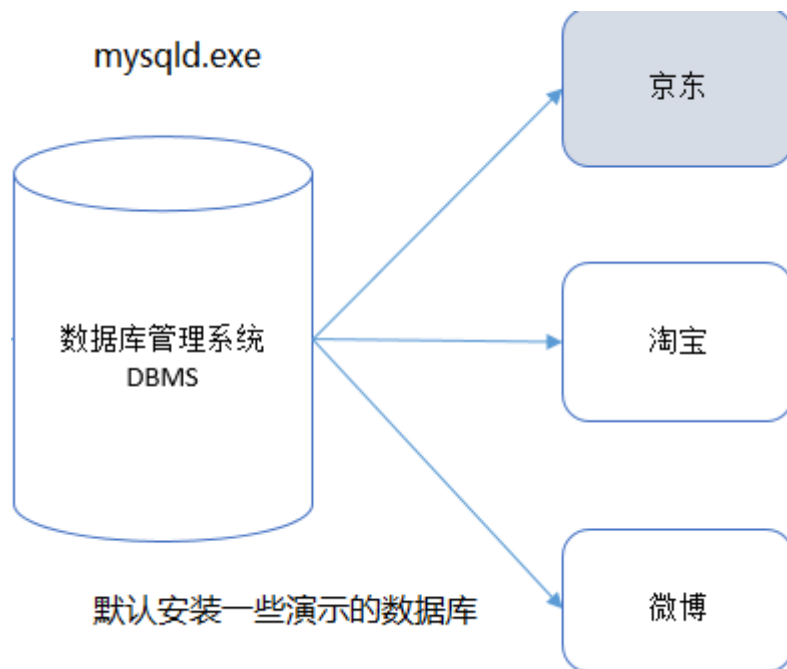
JDBC

学习目标

1. 能够使用DCL处理MySQL中的用户
2. 能够理解JDBC的概念
3. 能够使用Connection接口
4. 能够使用Statement接口
5. 能够使用ResultSet接口
6. 能够使用JDBC实现对单表数据增、删、改、查
7. 能够使用JDBC操作事务
8. 能够编写JDBC工具类
9. 能够完成JDBC实现登录案例

第1章 DCL

我们现在默认使用的都是root用户，超级管理员，拥有全部的权限。但是，一个公司里面的数据库服务器上可能同时运行着很多个项目的数据库。所以，我们应该可以根据不同的项目建立不同的用户，分配不同的权限来管理和维护数据库。



1.1 创建用户

`CREATE USER '用户名'@'主机名' IDENTIFIED BY '密码';` **关键字说明:**

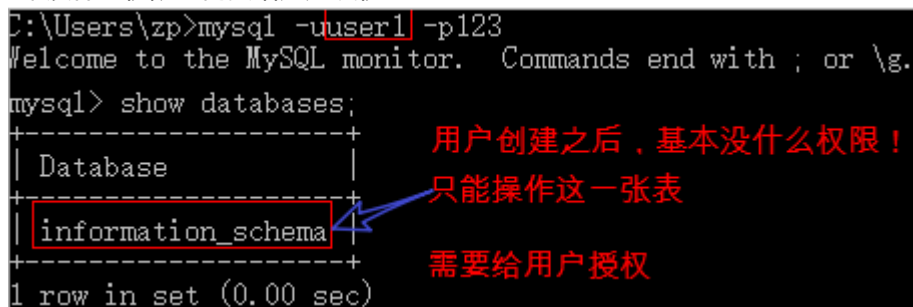
1. `用户名`: 将创建的用户名
2. `主机名`: 指定该用户在哪个主机上可以登陆，如果是本地用户可用localhost，如果想让该用户可以从任意远程主机登陆，可以使用通配符%
3. `密码`: 该用户的登陆密码，密码可以为空，如果为空则该用户不需要密码登陆服务器

具体操作:

```
-- user1用户只能在localhost这个IP登录mysql服务器
CREATE USER 'user1'@'localhost' IDENTIFIED BY '123';
-- user2用户可以在任何电脑上登录mysql服务器
CREATE USER 'user2'@'%' IDENTIFIED BY '123';
```

1.2 授权用户

用户创建之后，基本没什么权限！需要给用户授权



The screenshot shows a MySQL command prompt where user1 logs in and runs 'show databases;'. The output shows only 'information_schema'. Red text annotations state: '用户创建之后，基本没什么权限！' (After user creation, there are basically no permissions!), '只能操作这一张表' (Can only operate on this table), and '需要给用户授权' (Need to grant permissions to the user). A blue arrow points from the text to the 'information_schema' entry in the output.

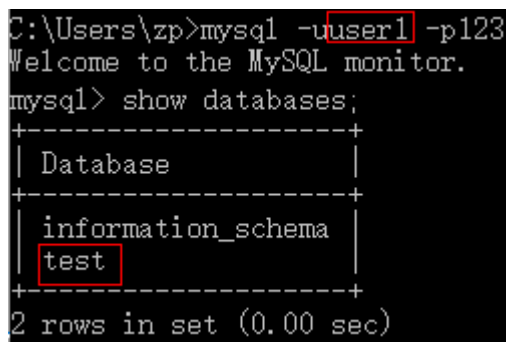
授权格式: GRANT 权限1, 权限2... ON 数据库名.表名 TO '用户名'@'主机名'; **关键字说明:**

1. `GRANT` 授权关键字
2. 授予用户的权限，如`SELECT`，`INSERT`，`UPDATE`等。如果要授予所有的权限则使用`ALL`
3. `数据库名.表名`：该用户可以操作哪个数据库的哪些表。如果要授予该用户对所有数据库和表的相应操作权限则可用*表示，如`*.*`
4. `'用户名'@'主机名'`：给哪个用户授权

具体操作:

1. 给user1用户分配对test这个数据库操作的权限

```
GRANT CREATE,ALTER,DROP,INSERT,UPDATE,DELETE,SELECT ON test.* TO 'user1'@'localhost';
```



The screenshot shows the same MySQL command prompt as before, but after running the GRANT command. The output of 'show databases;' now includes both 'information_schema' and 'test'. The 'test' entry is highlighted with a red box.

2. 给user2用户分配对所有数据库操作的权限

```
GRANT ALL ON *.* TO 'user2'@'%';
```

```
C:\Users\zp>mysql -uuser2 -p123
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| day21 |
| day22 |
| day23 |
| day24 |
| db1 |
| db3 |
| mysql |
| performance_schema |
| test |
+-----+
```

uer2可以操作所有数据库

1.3 撤销授权

REVOKE 权限1, 权限2... ON 数据库.表名 FROM '用户名'@'主机名';

具体操作:

- 撤销user1用户对test操作的权限

```
REVOKE ALL ON test.* FROM 'user1'@'localhost';
```

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| test |
+-----+
2 rows in set (0.00 sec)

mysql> REVOKE CREATE, ALTER, DROP ON test.* FROM 'user1'@'localhost';
Query OK, 0 rows affected (0.00 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
+-----+
1 row in set (0.00 sec)
```

取消授权

不能操作test数据库了

1.4 查看权限

SHOW GRANTS FOR '用户名'@'主机名'; 具体操作:

- 查看user1用户的权限

```
SHOW GRANTS FOR 'user1'@'localhost';
```



```
mysql> SHOW GRANTS FOR 'user1'@'localhost';
+-----+
| Grants for user1@localhost |
+-----+
| GRANT USAGE ON *.* TO 'user1'@'localhost' IDENTIFIED BY PASSWORD '*23AE809DDACAF96AF0FD78ED04B6A265E05AA257' |
| GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, ALTER ON `test`.* TO 'user1'@'localhost' | user1的权限 |
+-----+
2 rows in set (0.00 sec)
```

1.5 删除用户

`DROP USER '用户名'@'主机名';` 具体操作:

- 删除user2

```
DROP USER 'user2'@'%';
```

```
mysql> DROP USER 'user2'@'%';
Query OK, 0 rows affected (0.00 sec)

C:\Users\zp>mysql -uuser2 -p123
ERROR 1045 (28000): Access denied for user 'user2'@'localhost' (using password: YES)
没有这个账号了
```

1.6 修改用户密码

1.6.1 修改管理员密码

`mysqladmin -uroot -p password 新密码 -- 新密码不需要加上引号`

注意：需要在未登陆MySQL的情况下操作。

具体操作:

```
mysqladmin -uroot -p password 123456
输入老密码
```

```

新密码
C:\Users\zp>mysqladmin -uroot -p password 123456
Enter password: **** 输入老密码
C:\Users\zp>mysql -uroot -p123456 使用新密码登录
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 24
Server version: 5.5.49 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> 登录成功
```

1.6.2 修改普通用户密码

`set password for '用户名'@'主机名' = password('新密码');`

注意：需要在登陆MySQL的情况下操作。

具体操作：

```
`set password for 'user1'@'localhost' = password('666666');`
```

```
在MySQL登录的状态下                                新密码
mysql> set password for 'user1'@'localhost' = password('666666');
Query OK, 0 rows affected (0.00 sec)
C:\Users\zp>mysql -uuser1 -p123 老密码登录不了了
ERROR 1045 (28000): Access denied for user 'user1'@'localhost' (using password: YES)

C:\Users\zp>mysql -uuser1 -p666666 使用新密码登录成功了
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 26
Server version: 5.5.49 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> _
```

第2章 JDBC的概念

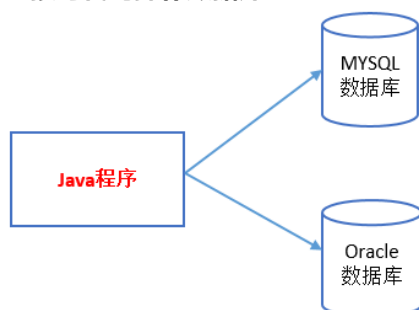
客户端操作MySQL数据库的方式

1. 使用第三方客户端来访问MySQL：SQLyog、Navicat、SQLWave、MyDB Studio、EMS SQL Manager for MySQL
2. 使用MySQL自带的命令行方式
3. 通过Java来访问MySQL数据库，今天要学习的内容

什么是JDBC：Java Data Base Connectivity (Java数据库连接) JDBC是Java访问数据库的 标准规范 JDBC的作用：JDBC是用于执行SQL语句的Java API (Java语言通过JDBC可以操作数据库)

第3章 JDBC的由来

1. 直接写代码操作数据库



直接写代码操作数据库存在以下问题

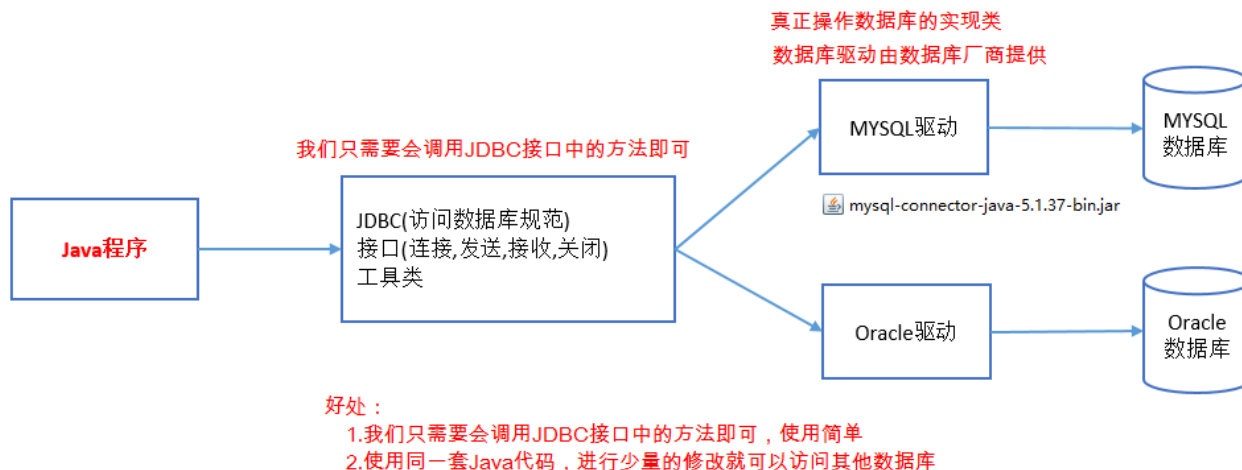
1. 不知道MySQL数据库的操作方式，解析方式
2. 代码繁琐，写起来麻烦
3. MySQL和Oracle数据库的操作方式和解析方式不同，每个数据库都要写一套代码
4. MySQL和Oracle数据库相互切换麻烦

直接写代码操作数据库存在的问题：

1. 不知道MySQL数据库的操作方式，解析方式

2. 代码繁琐，写起来麻烦
3. MySQL和Oracle等其他数据库的操作方式和解析方式不同，每个数据库都要写一套代码
4. MySQL和Oracle等其他数据库相互切换麻烦

2. **JDBC规范定义接口，具体的实现由各大数据库厂商来实现** JDBC是Java访问数据库的标准规范。真正怎么操作数据库还需要具体的实现类，也就是数据库驱动。每个数据库厂商根据自家数据库的通信格式编写好自己数据库的驱动。所以我们只需要会调用JDBC接口中的方法即可。数据库驱动由数据库厂商提供。



JDBC的好处：

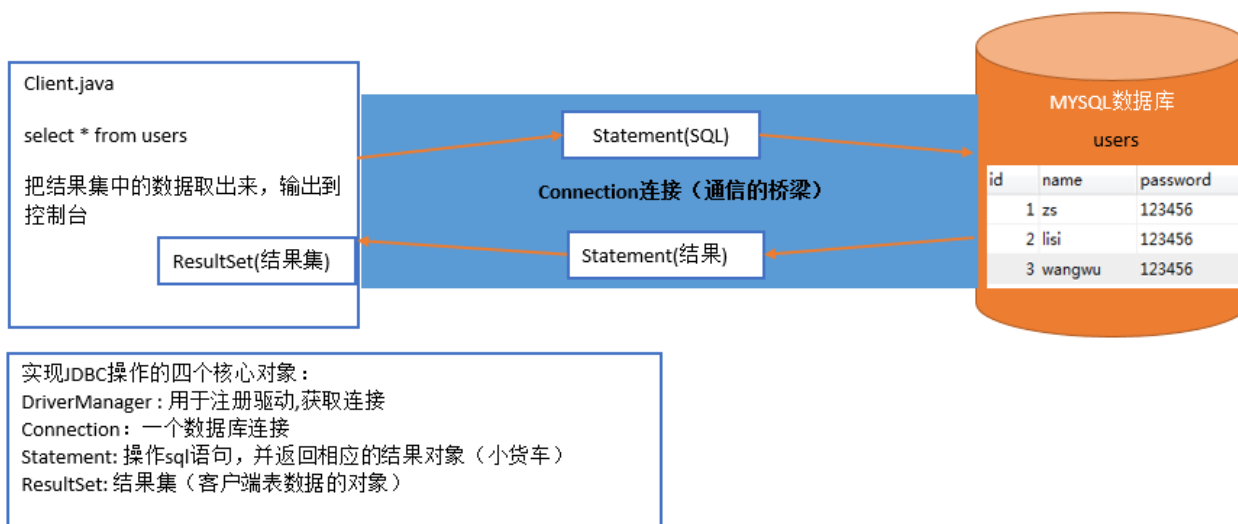
1. 我们只需要会调用JDBC接口中的方法即可，使用简单
2. 使用同一套Java代码，进行少量的修改就可以访问其他JDBC支持的数据库

JDBC会用到的包：

1. java.sql：JDBC访问数据库的基础包，在javaSE中的包。如：java.sql.Connection
2. javax.sql：JDBC访问数据库的扩展包
3. 数据库的驱动，各大数据库厂商来实现。如：MySQL的驱动：com.mysql.jdbc.Driver

JDBC四个核心对象 这几个类都是在java.sql包中

1. DriverManager: 用于注册驱动
2. Connection: 表示与数据库创建的连接
3. Statement: 执行SQL语句的对象
4. ResultSet: 结果集或一张虚拟表



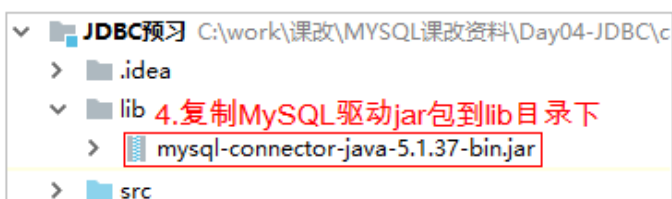
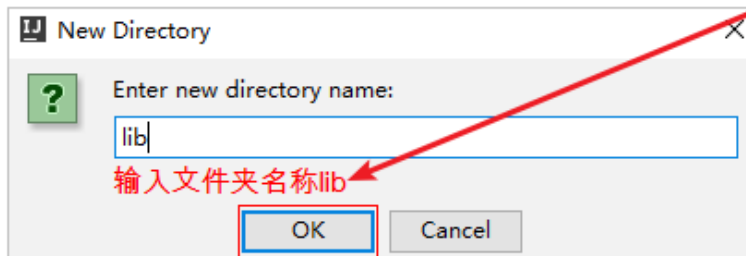
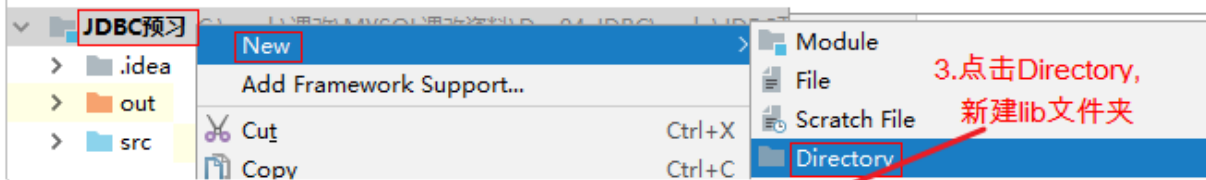
第4章 JDBC获取连接

`Connection` 表示Java程序与数据库之间的连接，只有拿到`Connection`才能操作数据库。

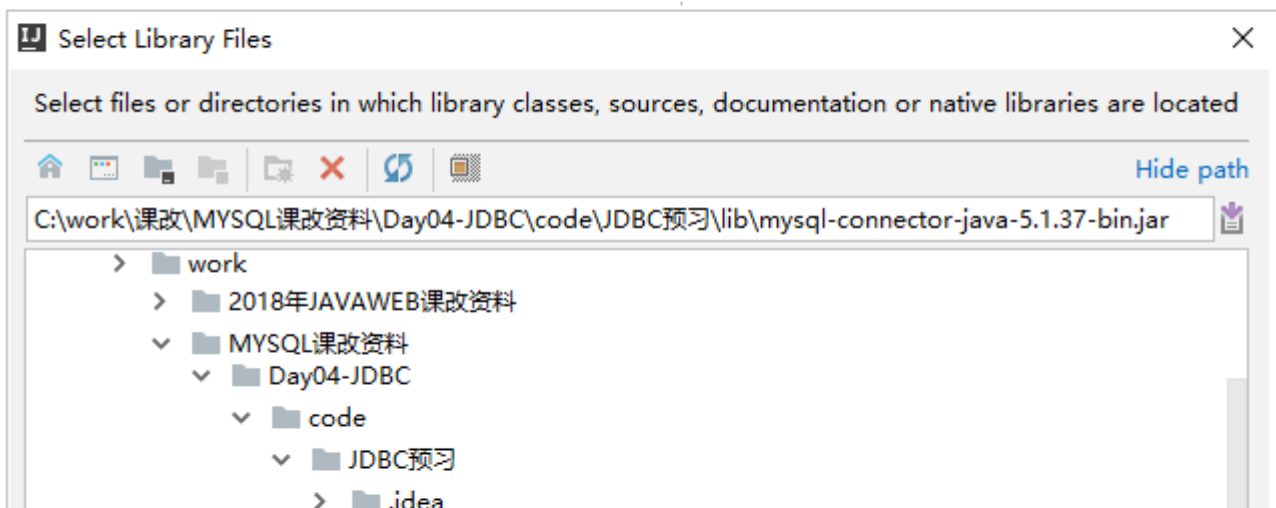
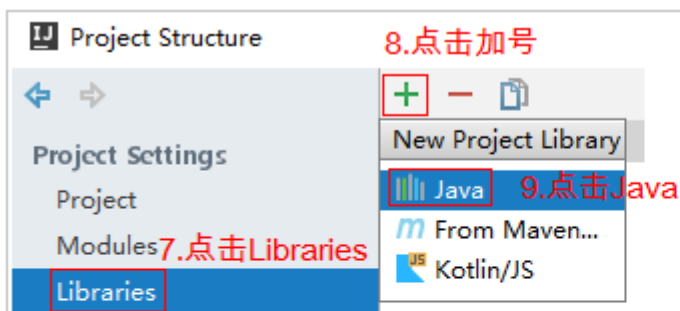
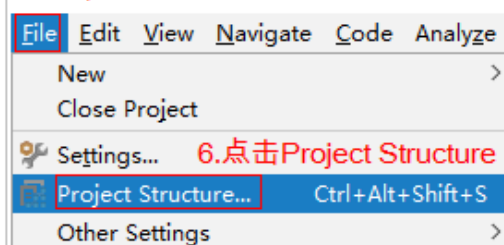
JDBC获取连接步骤 1.导入驱动jar包 2.注册驱动 3.获取连接

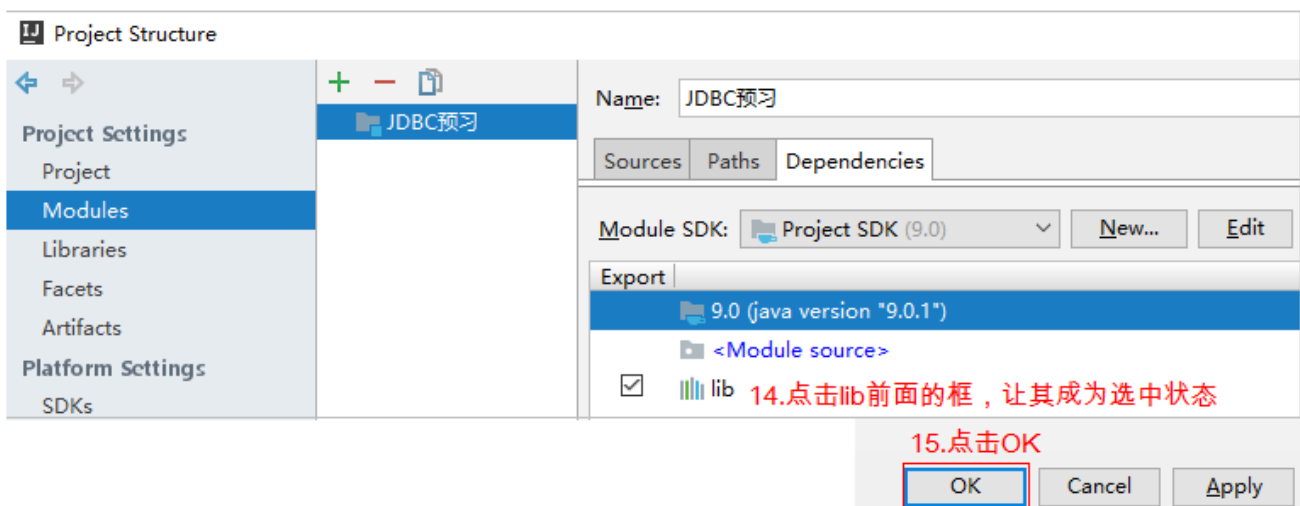
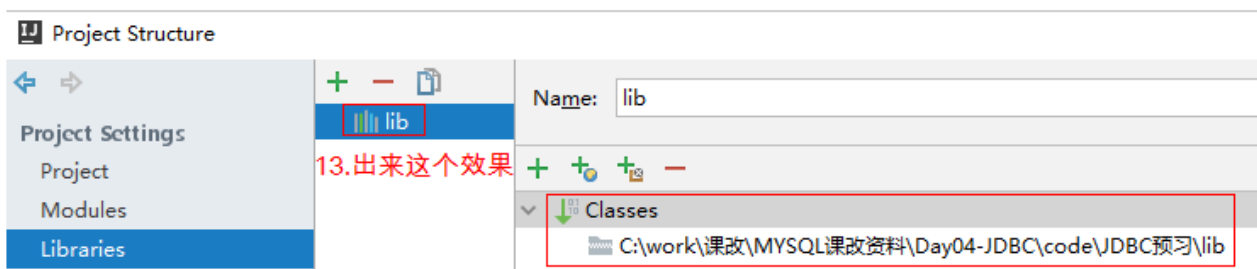
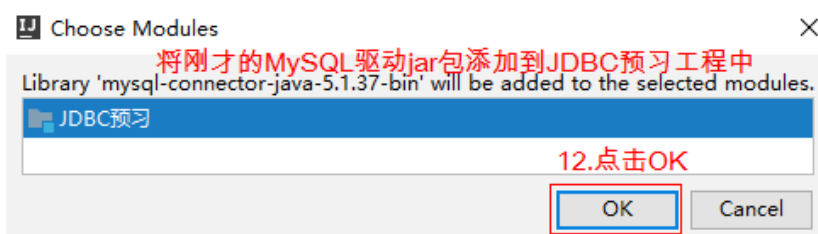
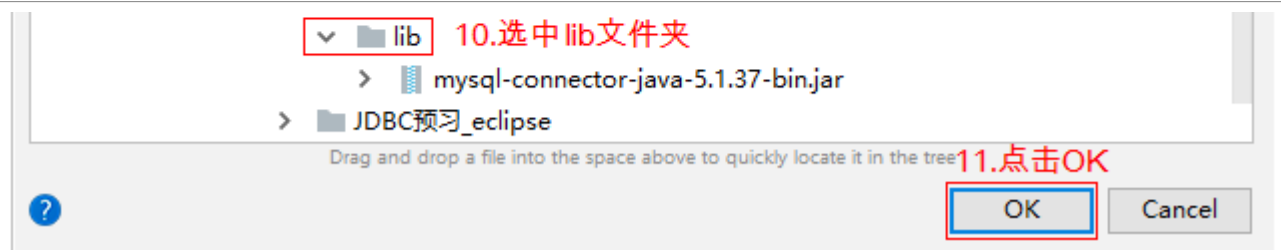
4.1 导入驱动jar包

1.工程上右键 2.点击New新建



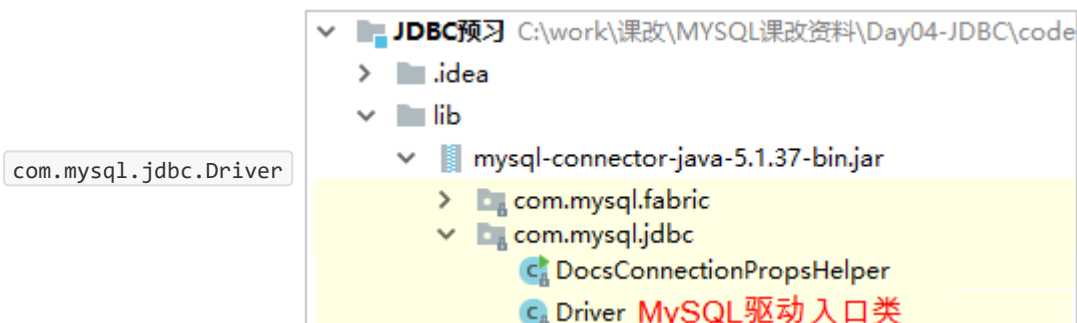
5.点击File





4.2 注册驱动

我们Java程序需要通过数据库驱动才能连接到数据库，因此需要注册驱动。MySQL的驱动的入口类是：



4.2.1 API介绍

`java.sql.DriverManager` 类用于注册驱动。提供如下方法注册驱动

```
static void registerDriver(Driver driver)
```

向 `DriverManager` 注册给定驱动程序。

4.2.2 使用步骤

1. `DriverManager.registerDriver(驱动对象)`; 传入对应参数即可

4.2.3 案例代码

```
public class Demo01 {  
    public static void main(String[] args) throws Exception {  
        // 注册驱动  
        DriverManager.registerDriver(new com.mysql.jdbc.Driver());  
    }  
}
```

通过查询 `com.mysql.jdbc.Driver` 源码，我们发现 `Driver` 类“主动”将自己进行注册

```
public class Driver extends NonRegisteringDriver implements java.sql.Driver {  
    static {  
        try {  
            // 自己自动注册  
            java.sql.DriverManager.registerDriver(new Driver());  
        } catch (SQLException E) {  
            throw new RuntimeException("Can't register driver!");  
        }  
    }  
    public Driver() throws SQLException {  
    }  
}
```

```
public class Demo01 {  
    public static void main(String[] args) throws Exception {  
        DriverManager.registerDriver(new com.mysql.jdbc.Driver());  
    }  
}
```

2. `Driver` 类创建好后又会注册一次
总共注册了2次

```
43 public class Driver extends NonRegisteringDriver implements java.sql.Driver {  
44     //  
45     // Register ourselves with the DriverManager  
46     // 1.第一次创建Driver对象走静态代码块，静态代码块中注册一次驱动  
47     static {  
48         try {  
49             java.sql.DriverManager.registerDriver(new Driver());  
50         } catch (SQLException E) {  
51             throw new RuntimeException("Can't register driver!");  
52         }  
53     }  
54  
55     public Driver() throws SQLException {  
56         // Required for Class.forName().newInstance()  
57     }  
58 }
```

注意：使用 `DriverManager.registerDriver(new com.mysql.jdbc.Driver());`，存在两方面不足

1. 硬编码，后期不易于程序扩展和维护



2. 驱动被注册两次

使用 `Class.forName("com.mysql.jdbc.Driver");` 加载驱动，这样驱动只会注册一次

```
public class Demo01 {
    public static void main(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver"); // 后期可以将"com.mysql.jdbc.Driver"字符串写在文
    }
}
```

件中。

演示： `Class.forName("包名.类名");` 会走这个类的静态代码块

```
9 public class Demo01 {
10     public static void main(String[] args) throws Exception {
11         Class.forName("com.itheima.demo01获取连接.Person");
12     }
13 }
14
15 class Person {
16     static {
17         System.out.println("我是Person静态代码块");
18     }
19 }
```

mo01

C:\develop\java\jdk-9.0.1\bin\java "-javaagent:C:\Program File

我是Person静态代码块

Process finished with exit code 0

通常开发我们使用 `Class.forName()` 加载驱动。 `Class.forName("com.mysql.jdbc.Driver");` 会走 `Driver` 类的静态代码块。在静态代码块中注册一次驱动。

```
9 public class Demo01 {
10     public static void main(String[] args) throws Exception {
11         Class.forName("com.mysql.jdbc.Driver");
12     }
13 }
14
15 class Driver extends NonRegisteringDriver implements java.sql.Driver {
16     //
17     // Register ourselves with the DriverManager
18     //
19     static {
20         try {
21             java.sql.DriverManager.registerDriver(new Driver());
22         } catch (SQLException E) {
23             throw new RuntimeException("Can't register driver!");
24         }
25     }
26
27     public Driver() throws SQLException {
28         // Required for Class.forName().newInstance()
29     }
30 }
```

总结：注册MySQL驱动使用 `Class.forName("com.mysql.jdbc.Driver");`

4.3 获取连接

4.3.1 API介绍

`java.sql.DriverManager` 类中有如下方法获取数据库连接



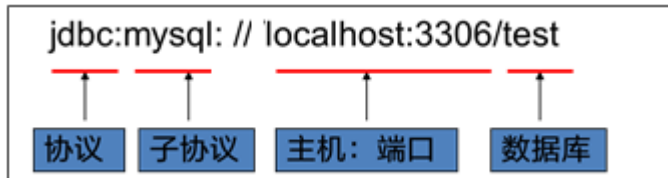
```
static Connection getConnection(String url, String user, String password)
```

连接到给定数据库 URL，并返回连接。

4.3.2 参数说明

1. `String url`：连接数据库的URL，用于说明连接数据库的位置
2. `String user`：数据库的账号
3. `String password`：数据库的密码

连接数据库的URL地址格式：`协议名:子协议://服务器名或IP地址:端口号/数据库名?参数=参数值`



MySQL写法: `jdbc:mysql://localhost:3306/day24`

如果是本地服务器，端口号是默认的3306，则可以简写：`jdbc:mysql:///day24`

4.3.3 注意事项

如果数据出现乱码需要加上参数: `?characterEncoding=utf8`，表示让数据库以UTF-8编码来处理数据。如：
`jdbc:mysql://localhost:3306/day24?characterEncoding=utf8`

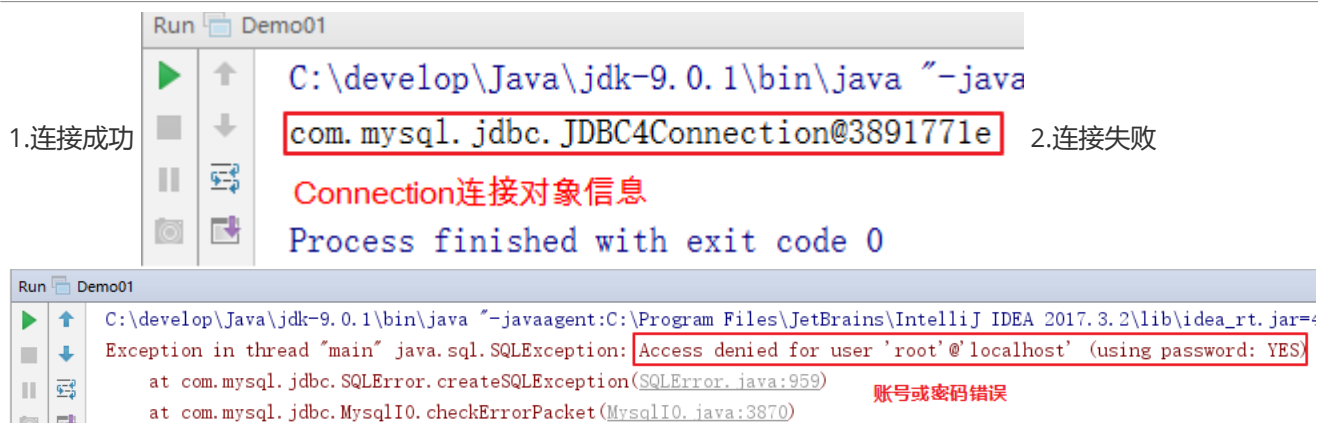
4.3.4 使用步骤

1. `DriverManager.getConnection(url, user, password)`; 传入对应参数即可

4.3.5 案例代码

```
public class Demo01 {  
    public static void main(String[] args) throws Exception {  
        Class.forName("com.mysql.jdbc.Driver");  
  
        // 连接到MySQL  
        // url: 连接数据库的URL  
        // user: 数据库的账号  
        // password: 数据库的密码  
        Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/day24",  
            "root", "root");  
        System.out.println(conn);  
    }  
}
```

4.3.6 案例效果



第5章 JDBC实现对单表数据增、删、改、查

我们要对数据库进行增、删、改、查，需要使用 `Statement` 对象来执行SQL语句。

5.1 准备数据

```
-- 创建分类表
CREATE TABLE category (
  cid INT PRIMARY KEY AUTO_INCREMENT,
  cname VARCHAR(100)
);
-- 初始化数据
INSERT INTO category (cname) VALUES('家电');
INSERT INTO category (cname) VALUES('服饰');
INSERT INTO category (cname) VALUES('化妆品');
```

5.2 JDBC实现对单表数据增、删、改

5.2.1 API介绍

5.2.1.1 获取Statement对象API介绍

在 `java.sql.Connection` 接口中有如下方法获取到 `Statement` 对象

```
Statement createStatement()
创建一个 Statement 对象来将 SQL 语句发送到数据库
```

5.2.1.2 Statement的API介绍

1. `boolean execute(String sql)`
此方法可以执行任意sql语句。返回boolean值，表示是否返回ResultSet结果集。仅当执行select语句，且有返回结果时返回true，其它语句都返回false;
2. `int executeUpdate(String sql)`
根据执行的DML (INSERT、UPDATE、DELETE) 语句，返回受影响的行数

3. `ResultSet executeQuery(String sql)`
根据查询语句返回结果集，只能执行SELECT语句

注意：在MySQL中，只要不是查询就是修改。 `executeUpdate`：用于执行增删改 `executeQuery`：用于执行查询

5.2.2 使用步骤

1. 注册驱动
2. 获取连接
3. 获取Statement对象
4. 使用Statement对象执行SQL语句
5. 释放资源

5.2.3 案例代码

```
public class Demo03 {  
    public static void main(String[] args) throws Exception {  
        Class.forName("com.mysql.jdbc.Driver");  
  
        Connection conn = DriverManager.getConnection("jdbc:mysql:///day24", "root", "root");  
        System.out.println(conn);  
  
        // String sql = "SELECT * FROM category;";  
        // 从连接中拿到一个Statement对象  
        Statement stmt = conn.createStatement();  
  
        // 1.插入记录  
        String sql = "INSERT INTO category (cname) VALUES ('手机');";  
        int i = stmt.executeUpdate(sql);  
        System.out.println("影响的行数:" + i);  
  
        // 2.修改记录  
        sql = "UPDATE category SET cname='汽车' WHERE cid=4;";  
        i = stmt.executeUpdate(sql);  
        System.out.println("影响的行数:" + i);  
  
        // 3.删除记录  
        sql = "DELETE FROM category WHERE cid=1;";  
        i = stmt.executeUpdate(sql);  
        System.out.println("影响的行数:" + i);  
  
        // 释放资源  
        stmt.close();  
        conn.close();  
    }  
}
```

5.2.4 案例效果

前

cid	cname
1	家电
2	服饰
3	化妆品

后

cid	cname
2	汽车
3	化妆品
4	手机

1.插入记录

2.修改记录

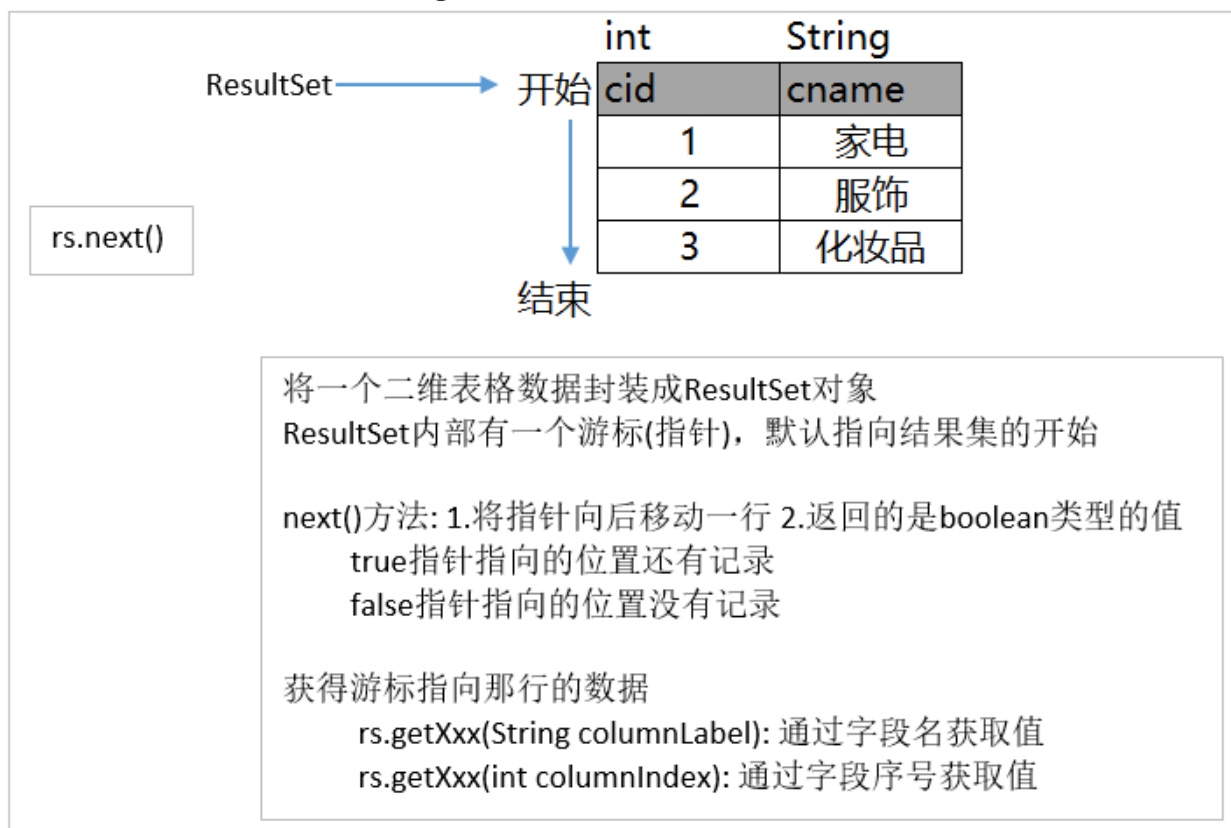
3.删除记录

5.3 JDBC实现对单表数据查询

`ResultSet` 用于保存执行查询SQL语句的结果。我们不能一次性取出所有的数据，需要一行一行的取出。

ResultSet的原理：

1. `ResultSet`内部有一个指针,刚开始记录开始位置
2. 调用`next`方法, `ResultSet`内部指针会移动到下一行数据
3. 我们可以通过`ResultSet`得到一行数据 `getXxx`得到某列数据



ResultSet获取数据的API 其实ResultSet获取数据的API是有规律的get后面加数据类型。我们统称 `getXXX()`

boolean	<code>getBoolean(String columnLabel)</code> 以 Java 编程语言中 boolean 的形式获取此 ResultSet 对象的当前行中指定列的值。
byte	<code>getByte(String columnLabel)</code> 以 Java 编程语言中 byte 的形式获取此 ResultSet 对象的当前行中指定列的值。
short	<code>getShort(String columnLabel)</code> 以 Java 编程语言中 short 的形式获取此 ResultSet 对象的当前行中指定列的值。
long	<code>getLong(String columnLabel)</code> 以 Java 编程语言中 long 的形式获取此 ResultSet 对象的当前行中指定列的值。
float	<code>getFloat(String columnLabel)</code> 以 Java 编程语言中 float 的形式获取此 ResultSet 对象的当前行中指定列的值。
double	<code>getDouble(String columnLabel)</code> 以 Java 编程语言中 double 的形式获取此 ResultSet 对象的当前行中指定列的值。
String	<code>getString(String columnLabel)</code> 以 Java 编程语言中 String 的形式获取此 ResultSet 对象的当前行中指定列的值。

使用JDBC查询数据库中的数据步骤

1. 注册驱动
2. 获取连接
3. 获取到Statement
4. 使用Statement执行SQL
5. ResultSet处理结果
6. 关闭资源

案例代码

```
public class Demo04 {  
    public static void main(String[] args) throws Exception {  
        Class.forName("com.mysql.jdbc.Driver");  
  
        Connection conn = DriverManager.getConnection("jdbc:mysql:///day24", "root", "root");  
        Statement stmt = conn.createStatement();  
  
        String sql = "SELECT * FROM category;";  
        ResultSet rs = stmt.executeQuery(sql);  
  
        // 内部有一个指针,只能取指针指向的那条记录  
        while (rs.next()) { // 指针移动一行,有数据才返回true  
            // 取出数据  
            int cid = rs.getInt("cid");  
            String cname = rs.getString("cname");  
  
            System.out.println(cid + " == " + cname);  
        }  
  
        // 关闭资源  
        rs.close();  
        stmt.close();  
        conn.close();  
    }  
}
```


注意：

1. 如果光标在第一行之前，使用rs.getXXX()获取列值，报错：Before start of result set
2. 如果光标在最后一行之后，使用rs.getXXX()获取列值，报错：After end of result set

数据库中数据

cid	cname
2	汽车
3	化妆品
4	手机

程序查询出数据

Console
develop\Java\j
2 == 汽车
3 == 化妆品
4 == 手机

案例效果

总结：其实我们使用JDBC操作数据库的步骤都是固定的。不同的地方是在编写SQL语句

1. 注册驱动
2. 获取连接
3. 获取到Statement
4. 使用Statement执行SQL
5. ResultSet处理结果
6. 关闭资源

第6章 JDBC事务

之前我们是使用MySQL的命令来操作事务。接下来我们使用JDBC来操作银行转账的事务。

6.1 准备数据

```
CREATE TABLE account (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    NAME VARCHAR(10),  
    balance DOUBLE  
);  
-- 添加数据  
INSERT INTO account (NAME, balance) VALUES ('张三', 1000), ('李四', 1000);
```

6.2 API介绍

Connection 接口中与事务有关的方法

1. `void setAutoCommit(boolean autoCommit) throws SQLException;`
false: 开启事务, true: 关闭事务
2. `void commit() throws SQLException;`
提交事务



3. `void rollback() throws SQLException;`
回滚事务

6.3 使用步骤

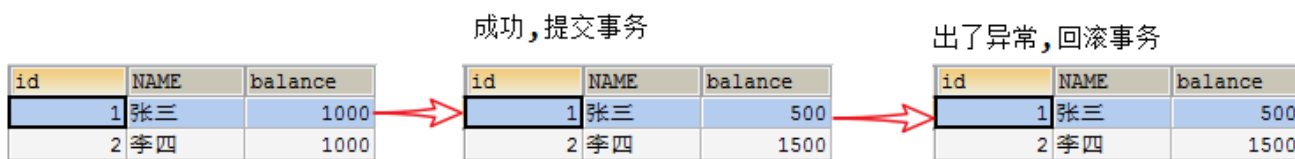
1. 注册驱动
2. 获取连接
3. 获取到Statement
4. 开启事务
5. 使用Statement执行SQL
6. 提交或回滚事务
7. 关闭资源

6.4 案例代码

```
public class Demo05 {  
    public static void main(String[] args) {  
        Connection conn = null;  
        try {  
            // 拿到连接  
            Class.forName("com.mysql.jdbc.Driver");  
            conn = DriverManager.getConnection("jdbc:mysql:///day24", "root", "root");  
  
            // 开启事务  
            conn.setAutoCommit(false);  
  
            Statement pstmt = conn.createStatement();  
  
            // 张三减500  
            String sql = "UPDATE account SET balance = balance - 500 WHERE id=1;";  
            pstmt.executeUpdate(sql);  
            // 模拟异常  
            // int i = 10 / 0;  
  
            // 李四加500  
            sql = "UPDATE account SET balance = balance + 500 WHERE id=2;";  
            pstmt.executeUpdate(sql);  
  
            pstmt.close();  
            // 成功,提交事务  
            System.out.println("成功,提交事务");  
            conn.commit();  
        } catch (Exception e) {  
            // 失败,回滚事务  
            try {  
                System.out.println("出了异常,回滚事务");  
                conn.rollback();  
            } catch (SQLException e1) {  
                e1.printStackTrace();  
            }  
        }  
    }  
}
```

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

6.5 案例效果



第7章 JDBC获取连接与关闭连接工具类实现

通过上面案例需求我们会发现每次去执行SQL语句都需要注册驱动, 获取连接, 得到Statement, 以及释放资源。发现很多重复的劳动, 我们可以将重复的代码定义到某个类的方法中。直接调用方法, 可以简化代码。那么我们接下来定义一个 `JDBCUtils` 类。把注册驱动, 获取连接, 得到Statement, 以及释放资源的代码放到这个类的方法中。以后直接调用方法即可。

7.1 编写JDBC工具类步骤

1. 将固定字符串定义为常量
2. 在静态代码块中注册驱动(只注册一次)
3. 提供一个获取连接的方法 `static Connection getConnection();`
4. 定义关闭资源的方法 `close(Connection conn, Statement stmt, ResultSet rs)`
5. 重载关闭方法 `close(Connection conn, Statement stmt)`

7.2 案例代码

JDBCUtils.java

```
public class JDBCUtils {  
    // 1. 将固定字符串定义为常量  
    private static final String DRIVER_CLASS = "com.mysql.jdbc.Driver";  
    private static final String URL = "jdbc:mysql:///day24";  
    private static final String USER = "root";  
    private static final String PASSWORD = "root";  
  
    // 2. 在静态代码块中注册驱动(只注册一次)  
    // 当这个类加载到内存的时候就走这个静态代码块, 再去触发Driver类中的静态代码块, 主动注册  
    static {  
        try {  
            Class.forName(DRIVER_CLASS);  
        } catch (ClassNotFoundException e) {}  
    }  
}
```



```
}

// 3.提供一个获取连接的方法static Connection getConneciton();
// 我们面向JDBC编程
public static Connection getConnection() throws SQLException {
    InputStream is = JDBCUtils.class.getResourceAsStream("/jdbc.properties");
    Properties pp = new Properties();
    pp.load(is);

    Connection conn = DriverManager.getConnection(URL, pp);
    return conn;
}

// 5.重载关闭方法close(Connection conn, Statement stmt)
public static void close(Connection conn, Statement stmt) {
    close(conn, stmt, null);
}

// 4.定义关闭资源的方法close(Connection conn, Statement stmt, ResultSet rs)
public static void close(Connection conn, Statement stmt, ResultSet rs) {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException e) {}
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException e) {}
    }

    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException e) {}
    }
}
}
```

Demo06.java

```
public class Demo06 {
    public static void main(String[] args) throws Exception {
        createTable();
        // addEmployee();
        // updateEmployee();
        // deleteEmployee();
    }

    // 删除员工
    public static void deleteEmployee() throws Exception {
        Connection conn = JDBCUtils.getConnection();
```



```
Statement stmt = conn.createStatement();

// 删除id为3的员工
String sql = "DELETE FROM employee WHERE id=3;";

int i = stmt.executeUpdate(sql);
System.out.println("影响的行数: " + i);

// stmt.close();
// conn.close();
// JDBCUtils.close(conn, stmt, null);
// JDBCUtils.close(conn, stmt);
}

// 修改员工
public static void updateEmployee() throws Exception {
    Connection conn = JDBCUtils.getConnection();
    Statement stmt = conn.createStatement();

    // 将id为3的员工姓名改成田七，地址改成天津
    String sql = "UPDATE employee SET address='天津', name='田七' WHERE id=3;";

    int i = stmt.executeUpdate(sql);
    System.out.println("影响的行数: " + i);

    // stmt.close();
    // conn.close();
    // JDBCUtils.close(conn, stmt, null);
    // JDBCUtils.close(conn, stmt);
}

// 定义添加员工
public static void addEmployee() throws Exception {
    Connection conn = JDBCUtils.getConnection();
    Statement stmt = conn.createStatement();

    // 添加4个员工
    String sql = "INSERT INTO employee VALUES (NULL, '张三4', 20, '北京'),"
        + " (NULL, '李四4', 21, '南京'),"
        + " (NULL, '王五4', 18, '东京'),"
        + " (NULL, '赵六4', 17, '西安');";

    int i = stmt.executeUpdate(sql);
    System.out.println("影响的行数: " + i);

    // stmt.close();
    // conn.close();
    // JDBCUtils.close(conn, stmt, null);
    // JDBCUtils.close(conn, stmt);
}

// 创建表

public static void createTable() throws Exception {
```

```
Connection conn = JDBCUtils.getConnection();
Statement stmt = conn.createStatement();

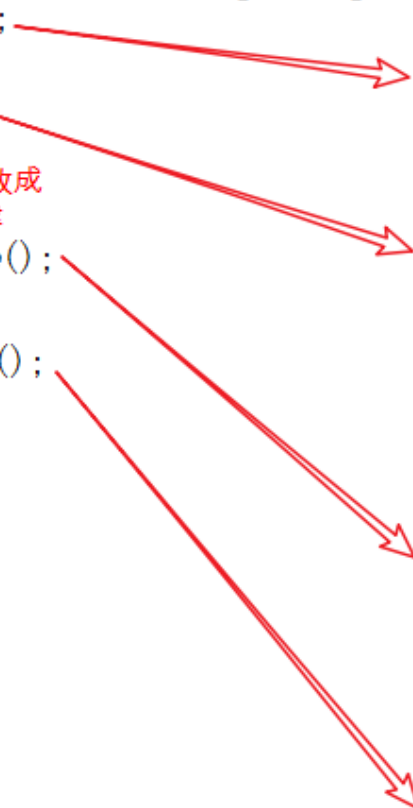
String sql = "CREATE TABLE IF NOT EXISTS employee ("
    + " id INT PRIMARY KEY AUTO_INCREMENT,"
    + " name VARCHAR(20) UNIQUE NOT NULL,"
    + " age INT,"
    + " address VARCHAR(50)"
    + ");";

int i = stmt.executeUpdate(sql);
System.out.println("ok");

// stmt.close();
// conn.close();
// JDBCUtils.close(conn, stmt, null);
// JDBCUtils.close(conn, stmt);
}
```

7.3 案例效果

```
public static void main(String[] args) throws Exception {
    createTable();
    // 添加4个员工
    addEmployee();
    // 将id为3的员工姓名改成田七，地址改成天津
    updateEmployee();
    // 删除id为3的员工
    deleteEmployee();
}
```



id	name	age	address
(Auto)	(NULL)	(NULL)	(NULL)

id	name	age	address
1	张三4	20	北京
2	李四4	21	南京
3	王五4	18	东京
4	赵六4	17	西安

id	name	age	address
1	张三4	20	北京
2	李四4	21	南京
3	田七	18	天津
4	赵六4	17	西安

id	name	age	address
1	张三4	20	北京
2	李四4	21	南京
4	赵六4	17	西安

第8章 JDBC实现登录案例

8.1 案例需求

模拟用户输入账号和密码登录网站

8.2 案例效果

1. 输入正确的账号，密码，显示登录成功

请输入账号：
admin 输入正确的账号，密码
请输入密码：
123 登录成功
欢迎您,admin

2. 输入错误的账号，密码，显示登录失败

请输入账号：
admin 输入错误的账号或密码
请输入密码：
aaa 显示登录失败
账号或密码错误...

8.3 案例分析

1. 使用数据库保存用户的账号和密码
2. 让用户输入账号和密码
3. 使用SQL根据用户的账号和密码去数据库查询数据
4. 如果查询到数据，说明登录成功
5. 如果查询不到数据，说明登录失败

8.4 实现步骤

1. 创建一个用户表保存用户的账号和密码，并添加一些数据，SQL语句如下：

```
CREATE TABLE USER (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    NAME VARCHAR(50),  
    PASSWORD VARCHAR(50)  
);  
INSERT INTO USER (NAME, PASSWORD) VALUES('admin', '123'), ('test', '123'), ('gm', '123');
```

2. 编写代码让用户输入账号和密码

```
public class Demo07 {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        System.out.println("请输入账号: ");  
        String name = sc.nextLine();  
        System.out.println("请输入密码: ");  
        String password = sc.nextLine();  
    }  
}
```



1. 使用SQL根据用户的账号和密码去数据库查询数据

```
public class Demo07 {  
    public static void main(String[] args) throws Exception {  
        // 让用户输入账号和密码  
        Scanner sc = new Scanner(System.in);  
        System.out.println("请输入账号: ");  
        String name = sc.nextLine();  
        System.out.println("请输入密码: ");  
        String password = sc.nextLine();  
  
        // 使用SQL根据用户的账号和密码去数据库查询数据  
        Connection conn = JDBCUtils.getConnection();  
        Statement stmt = conn.createStatement();  
        String sql = "SELECT * FROM user WHERE name='" + name + "' AND password='" + password +  
            "';";  
    }  
}
```

2. 如果查询到数据，说明登录成功，如果查询不到数据，说明登录失败

```
public class Demo07 {  
    public static void main(String[] args) throws Exception {  
        // 让用户输入账号和密码  
        Scanner sc = new Scanner(System.in);  
        System.out.println("请输入账号: ");  
        String name = sc.nextLine();  
        System.out.println("请输入密码: ");  
        String password = sc.nextLine();  
  
        // 使用SQL根据用户的账号和密码去数据库查询数据  
        Connection conn = JDBCUtils.getConnection();  
        Statement stmt = conn.createStatement();  
        String sql = "SELECT * FROM user WHERE name='" + name + "' AND password='" + password +  
            "';";  
  
        // 如果查询到数据，说明登录成功，如果查询不到数据，说明登录失败  
        ResultSet rs = stmt.executeQuery(sql);  
  
        if (rs.next()) {  
            //能进来查询到了数据。  
            String name2 = rs.getString("name");  
            System.out.println("欢迎您," + name2);  
        } else {  
            //查询不到数据，说明登录失败  
            System.out.println("账号或密码错误...");  
        }  
  
        JDBCUtils.close(conn, stmt, rs);  
    }  
}
```