数据库编程实验

环境

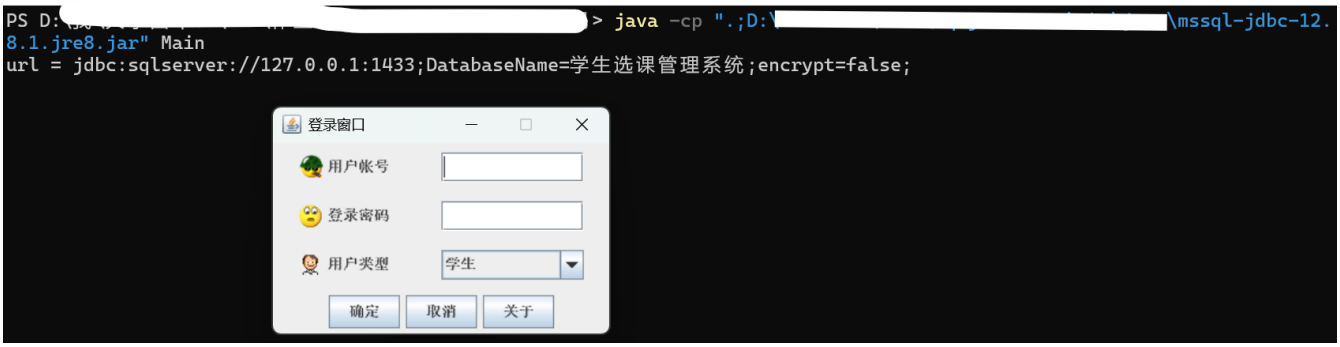
SQL SERVER + JDBC。

实验内容

使用提供的学生成绩管理系统数据库。

实验过程

1. 解压源码，在其目录下执行 `java Main`，但是此处多次尝试显示无模块，配置环境变量也显示无效，可能需要重启，因此选择直接把依赖包放到java命令参数里面，同时显示不支持加密因此选择在url中添加参数 `encrypt=false;` 解决，最终执行命令为：`java -cp ".;<path_to_jdbc_jar>" Main`，成功运行，通过在数据库查询的账号密码成功登录：



2. 审计 `LoginFrame.java` 源码：首先获取数据库的连接，然后根据获取的用户输入，先判断用户类型（学生 or 教师 or 管理员），再将输入的账号密码拼接到预设的SQL语句中（此处是**SQL注入**的根源）进行查询操

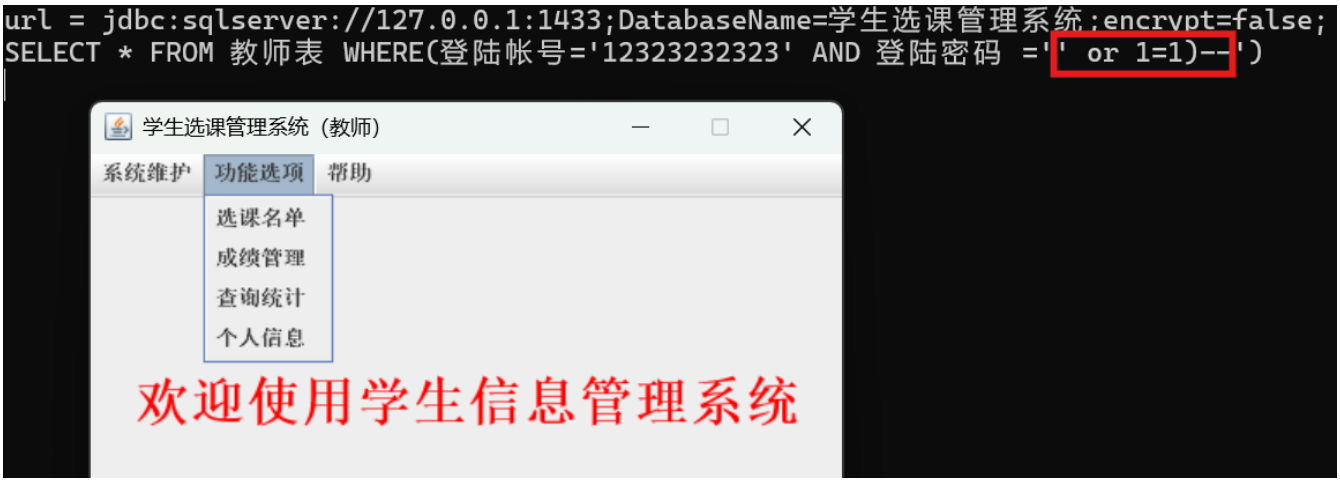
作，而后由拼接好的SQL语句，根据账号密码在对应表中查询，如果返回的查询结果非空（当前数据库对应表中有至少一条数据的账号密码与输入相同），即账号密码正确，置 `login` 为1标志登录状态，否则提示错误，结束验证过程（当前方法）；

```
// 核心登录逻辑代码
private void loginDispose()
    {
        //加载驱动程序以连接数据库
        try
        {
            Class.forName(DataBaseInfo.drive);
            loginConnection =
DriverManager.getConnection(DataBaseInfo.url,DataBaseInfo.username,DataBaseInfo.
password);
        }
        //捕获加载驱动程序异常
        catch ( ClassNotFoundException cnfex )
        {
            cnfex.printStackTrace();
            JOptionPane.showMessageDialog (LoginFrame.this, cnfex,
                                           "学生选课管理系统",
JOptionPane.WARNING_MESSAGE );
            System.exit( 1 );     // terminate program
        }
        //捕获连接数据库异常
        catch ( SQLException sqlex )
        {
            sqlex.printStackTrace();
            System.out.println(DataBaseInfo.url);
            JOptionPane.showMessageDialog (LoginFrame.this, "无法连接到SQL SERVER
, \n请确认SQL SERVER是否运行\n或数据源设置是否正确！ ",
                                           "学生选课管理系统",
JOptionPane.WARNING_MESSAGE );
            System.exit( 1 );  // terminate program
        }


        try
        {
            String loginQuery;
            String loginUserName =  myTextField.getText();
            String loginPassword = new String(passwordField.getPassword());
            if(myTextField.getText().equals( "" ))
            {
                JOptionPane.showMessageDialog( LoginFrame.this, "用户名必须为字
母、数字和、汉字\n及其组合，不允许为空格键。",
                                               "登陆", JOptionPane.WARNING_MESSAGE
);
                //setTitle( "无记录显示" );
                return;
            }
            if(selectedItem.equals("教师"))
                loginQuery = "SELECT * FROM 教师表 WHERE(登陆帐号='" +
loginUserName  + "' AND 登陆密码 ='" + loginPassword + "')";
            else if(selectedItem.equals("管理员"))
                loginQuery = "SELECT * FROM 管理员 WHERE(用户名='" +
```

```
loginUserName + "' AND 密码 ='" + loginPassword + "')";
            else //(selectedItem.equals("学生"))
                loginQuery = "SELECT * FROM 学生基本信息表 WHERE(学号='" +
loginUserName + "' AND 密码 ='" + loginPassword + "')";
            loginStatement = loginConnection.createStatement();
            System.out.println(loginQuery);  // XD
            loginResultSet = loginStatement.executeQuery( loginQuery );
            boolean Records = loginResultSet.next();
            if ( ! Records )
            {
                JOptionPane.showMessageDialog(LoginFrame.this, "没有此用户或密码
错误" );
                return;
            }
            else
            {
                login = 1 ;
            }
            loginConnection.close();
        }
        catch(SQLException sqlex)
        {
            //sqlex.printStackTrace();
            JOptionPane.showMessageDialog (LoginFrame.this, sqlex,
                                           "学生选课管理系统",
JOptionPane.WARNING_MESSAGE );
        }
    }
```

3. SQL注入完成无账号密码登录（要求为教师，实际所有类型都可以随意登录），使用的账号是随意输的，密码部分为框选部分，核心思路为利用永真式绕过验证，而后通过补全和注释的方法保证SQL语句没有语法错误即可。



审计SQL语句后，构造该payload的过程具体为：

1. 利用 `or 1=1` 这个永远为真的条件使得查询一定有结果，从而实现随意登录；
2. 显然密码在拼接的后面也是接近结尾的地方，所以在密码输入处构造；
3. 拼接的语句将每个输入用 `'` 包裹，因此要先输入一个 `'` 将前面已经在语句里面的 `'` 抵消掉；
4. 然后就可以写入上面的永真式绕过验证；
5. 完成后可以看到整个条件是被包在一个 `()` 中的，所以写上一个 `)` 进行补全；

6. 最后，后面还有原来拼接使用的 `'` 和 `)`，因此利用注释符将后面的内容全部注释掉，就可以构造出最终的 `payload`。

反思SQL注入预防

JDBC中可以使用 `PreparedStatement` 配合 `?` 占位预防，其原理是对SQL语句的预编译，即提前编译SQL语句使得不管传入的内容是什么都被当做一份数据，而非可执行的一部分语句。 最重要的是，不要相信任何来自用户的输入，这样才能尽可能的防止此类漏洞的产生。