Java MySQL系列课程08

------ MySQL安全之SQL注入式错误

本文档包含以下内容：

1. SQL注入简介
2. PrepareStatement预防SQL注入
3. 转义函数

#### 前言：

**“你了解SQL注入式错误吗？”**这种问题在考察MySQL的时候还是经常会被问到的，在日常开发过程中，我们一般更注重功能的实现，SQL安全的问题考虑的还是比较少。但是如果 SQL 语句使用不当，将会给应用系统造成很大的安全隐患，其中最重要的隐患就是 SQL注入式错误。本章主要讨论 MySQL下的 SQL 注入式错误以及相应的防范措施。

#### 一 SQL注入简介：

结构化查询语言（SQL）是一种用来和数据库交互的文本语言。SQL Injection 就是利用非法的SQL拼接，从而达到入侵数据库的目的。 它的产生主要是由于程序对用户输入的数据没有进行严格的过滤，导致非法的数据库SQL操作语句的执行。

SQL 注入（SQL Injection）攻击具有很大的危害，攻击者可以利用它读取、修改或者删除数据库内的数据，获取数据库中的用户名和密码等敏感信息，甚至可以获得数据库管理员的权限，而且SQL Injection 也很难防范，一般的防火墙也无法拦截 SQL Injection 攻击。

**用Java JDBC编程，模拟用户登录操作，访问user表的用户数据**，如下：



**一般用户登录时，输入的用户名和密码都正确，才能登录成功，否则登录失败**，如下代码组装好SQL语句进行登录操作：

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** Exception{    Class.*forName*("com.mysql.jdbc.Driver");  String url = "jdbc:mysql://localhost:3306/chat";  Connection con = DriverManager.*getConnection*(url, "root", "111111");  Statement statement = con.createStatement();  String sql = String.*format*("select \* from user where name='%s'"  + "and pwd='%s'", **"test", "123456"**); //模拟拼接SQL语句  ResultSet rs = statement.executeQuery(sql);  **while**(rs.next()) {  System.***out***.println("登录成功！");  **return**;  }  System.***out***.println("登录失败！");  } |

**上面的代码写了正确的用户名和密码，运行结果显示登录成功！**

**输入错误的密码，但是用户名输入如下，依然可以登录成功：**

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** Exception{    Class.*forName*("com.mysql.jdbc.Driver");  String url = "jdbc:mysql://localhost:3306/chat";  Connection con = DriverManager.*getConnection*(url, "root", "111111");  Statement statement = con.createStatement();  String sql = String.*format*("select \* from user where name='%s'"  + "and pwd='%s'", **"test'or'1=1"**, **"1234"**); //模拟拼接SQL语句  ResultSet rs = statement.executeQuery(sql);  **while**(rs.next()) {  System.***out***.println("登录成功！");  **return**;  }  System.***out***.println("登录失败！");  } |

上面代码中，密码输入是错误的，但是用户名是：**"test'or'1=1"**，那么组装好的SQL语句就成为：

**select \* from user where name=’test’ or ‘1=1’ and pwd=’1234’;**

上面的SQL语句就表示用户名或者密码任意一个正确都能登录，也就是根本不需要密码，用户名正确就行，这是非常危险的。

同样，上面的用户名和密码还可以改成**"test'#"，密码错误**，也能登录成功，因为#在MySQL的SQL语句中，表示注释，**select \* from user where name=’test’# or ‘1=1’ and pwd=’1234’;**相当于# or ‘1=1’ and pwd=’1234’全部成为注释语句了，只要用户名为test，就能登录成功。

对于上面代码演示的 SQL 注入错误，存在严重的安全隐患，后果可想而知，轻则获得数据信息，重则可以将数据进行非法更改。那么对这种情况有没有防范措施呢？请看下面的内容描述。

#### **二 PrepareStatement预防SQL注入：**

Java JDBC提供的PrepareStatement是带有SQL预编译特性的，通过通配符代表SQL语句中的参数，有效的防止了常见的SQL注入式错误，上面的代码修改如下：

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** Exception{    Class.*forName*("com.mysql.jdbc.Driver");  String url = "jdbc:mysql://localhost:3306/chat";  Connection con = DriverManager.*getConnection*(url, "root", "111111");  PreparedStatement statement = con.prepareStatement("select \* from user where name=? and pwd=?");  statement.setString(1, "test");  statement.setString(2, "123456");  ResultSet rs = statement.executeQuery();  **while**(rs.next()) {  System.***out***.println("登录成功！");  **return**;  }  System.***out***.println("登录失败！");  } |

**上面的代码无论如何，都不会出现第一部分的SQL注入问题！**

Java后面学习的MyBatis框架，面试官经常爱问这样的问题**“MyBaits能防止SQL注入错误吗？”，**当然可以，通过Log4j日志可以打印出来MyBatis在执行过程中，底层也是通过PrepareStatement来操作SQL的。