SQLi in Java

SQL注入是Web安全老生常谈的一个话题，这里也没什么新的东西。

由于个人的一些需求，整理了一些Java中可能产生SQL注入的操作，做代码审计的时候可以关注一下。

不足或者错误的地方也麻烦指出。

SQL操作

JDBC

主要的操作就在`java.sql.Statement`这个接口下

- `java.sql.Statement.executeQuery`
- `java.sql.Statement.executeUpdate`
- `java.sql.Statement.execute`
- `java.sql.Statement.addBatch`
- `java.sql.Statement.executeUpdate`

应该算是一个最基础的SQL操作库。

SpringJDBC

主要对应`org.springframework.jdbc.core.JdbcTemplate`这个类下的操作

- `org.springframework.jdbc.core.JdbcTemplate.update`
- `org.springframework.jdbc.core.JdbcTemplate.execute`
- `org.springframework.jdbc.core.JdbcTemplate.query`
- `org.springframework.jdbc.core.JdbcTemplate.queryForObject`
- `org.springframework.jdbc.core.JdbcTemplate.queryForRowSet`
- `org.springframework.jdbc.core.JdbcTemplate.queryForList`
- `org.springframework.jdbc.core.JdbcTemplate.queryForMap`
- `org.springframework.jdbc.core.JdbcTemplate.batchUpdate`

里面有很多重载，例如

```
jdbcTemplate.queryForObject("select count(*) from user where name = '" + input+
"'");
jdbcTemplate.queryForObject("select count(*) from user where name = ?",
Integer.class, input);
```

其实第二种方式就是SpringJDBC的预编译，只要将值传入，就可以以预编译的方式执行SQL语句。

但假如第一个变量sql本身受到污染，也还是会存在SQL注入。

JPA

我在网上找了半天貌似只找到一个执行原生SQL的操作，而且使用概率极低

- `javax.persistence.EntityManager.createNativeQuery`

其余常规的用法貌似都是默认预编译的形式？求了解的大佬说一下。

Mybatis

算是使用频率比较高的一个现代化SQL操作库

有XML和注解两种形式

XML

这里以SpringBoot为例，XML的文件配置可以从`application`配置中获取

```
mybatis.mapper-locations: classpath:mapper/*.xml
```

也就是对应着mapper文件夹下的所有XML文件。

```
<mapper namespace="com.kingkk.sql.Dao.UserDao">
    <select id="findUserByName" parameterType="String"
resultType="com.kingkk.sql.Entity.User">
        SELECT * FROM user WHERE username = '${username}'
    </select>
</mapper>
```

如上的xml就表示这个mapper对应着`com.kingkk.sql.Dao.UserDao`这个接口下的`findUserByName`所要执行的SQL操作

注解

还有一种更方便的注解方式，可以直接注解在`UserDao`这个接口上

```
@Select("SELECT * FROM user WHERE username = '${username}'")
User findUserByName(@Param("username") String username);
```

占位符

在Mybatis中有两种占位符

- `#{}`为预编译占位符，这种情况不会存在SQL注入
- `${}`为字符串拼接占位符，本质就是字符串拼接，会存在SQL注入风险

所以审计时，可以寻找`${}`占位的变量，依次溯源。

Hibernate

主要对应`org.hibernate.Session`这个接口下的操作

- `org.hibernate.Session.createQuery`
- `org.hibernate.Session.createSQLQuery`
- `org.hibernate.Session.createNativeQuery`

Hibernate主要的操作对象其实不是SQL语句，而是HQL，但也可以执行原生SQL。

至于HQL也是可以注入的，具体的注入方法这里也就不讨论了。

为什么还要字符串拼接?

预编译算是一种能从根源上杜绝SQL注入的方式，而且几乎所有的库都支持，并且默认以预编译的方式执行SQL语句，但是为什么还是有那么多字符串的拼接?

整理了下一些常见的情况，也是可以关注的一些点。

开发基础太差

这种属于低级错误，例如之前的Mybatis，开发人员并不知道`${}`和`#{}`的区别，导致的乱用。

这种程序员写的程序可以放心不止SQL注入那么简单，但这种人属于少部分情况。

预编译姿势不对

例如使用like的时候，就经常容易出现一些SQL注入的情况。

```
select count(*) from user where username like '%?%'
```

原因是开发人员发现使用like的时候，如上的预编译方式会报错，但其实查阅官方手册之后，可以发现是可以通过正确的姿势进行预编译的。

```
select count(*) from user where username like concat('%',?,'%')
```

无法预编译

对于列表和表名，是不支持预编译的，也是产生字符串拼接最多的一种情况。

对于这种确实无法预编译的情况，首先推荐通过白名单的形式，限制传入的列名/表名的范围。

并且用`String.equals`、`List.contains`、`switch`等方式进行限制内容。

假如用的是`String.contains()`、`String.startsWith()`、`String.endsWith()`就依旧还是会被各种绕过。

至于无法限制范围的，就要经过正确的转义了，这里又是一个大坑。。