MyBatis 如何防止 SQL 注入

[摘自] mybatis 防止 sql 注入 - chaoge

SQL 注入是一种**代码注入技术**，用于攻击数据驱动的应用，**恶意的 SQL 语句**被插入到执行的实体字段中（例如，为了转储数据库内容给**攻击者**）。[摘自] SQL injection - Wikipedia

SQL 注入，大家都不陌生，是一种常见的攻击方式。**攻击者**在界面的表单信息或 URL 上输入一些奇怪的 SQL 片段（例如“or '1'='1'”这样的语句），有可能入侵**参数检验不足**的应用程序。所以，在我们的应用中需要做一些工作，来防备这样的攻击方式。在一些安全性要求很高的应用中(比如银行软件)，经常使用将 **SQL 语句**全部替换为**存储过程**这样的方式，来防止 SQL 注入。这当然是**一种很安全的方式**，但我们平时开发中，可能不需要这种死板的方式。

MyBatis 框架作为一款半自动化的持久层框架，其 SQL 语句都要我们自己手动编写，这个时候当然需要防止 **SQL 注入**。其实，MyBatis 的 SQL 是一个具有“**输入+输出**”的功能，类似于函数的结构，如下：

```
<select id="getBlogById" resultType="Blog" parameterType="int">
    SELECT id,title,author,content
    FROM blog
    WHERE id=#{id}
</select>
```

这里，**parameterType** 表示了**输入**的参数类型，**resultType** 表示了**输出**的参数类型。回应上文，如果我们想防止 SQL 注入，理所当然地要在**输入参数**上下功夫。上面代码中黄色高亮即输入参数在 SQL 中拼接的部分，传入参数后，打印出执行的 SQL 语句，会看到 SQL 是这样的：

```
SELECT id,title,author,content FROM blog WHERE id = ?
```

不管输入什么参数，打印出的 SQL 都是这样的。这是因为 **MyBatis 启用了预编译功能**，在 SQL 执行前，会先将上面的 SQL 发送给数据库进行编译；执行时，直接使用**编译好的 SQL**，替换**占位符**“**?**”就可以了。因为 **SQL 注入**只能对**编译过程**起作用，所以这样的方式就很好地**避免了 SQL 注入的问题**。

【底层实现原理】MyBatis 是如何做到 **SQL 预编译**的呢？其实在框架底层，是 JDBC 中的 **PreparedStatement** 类在起作用，PreparedStatement 是我们很熟悉的 Statement 的子类，它的对象包含了**编译好的 SQL 语句**。这种**“准备好”的方式**不仅能提高**安全性**，而且在多次执行**同一个 SQL** 时，能够提高**效率**。**原因**是 SQL 已编译好，再次执行时无需再编译。

话说回来，是否我们使用 MyBatis 就**一定可以防止 SQL 注入**呢？**当然不是**，请看下面的代码：

```
<select id="getBlogById" resultType="Blog" parameterType="int">
    SELECT id,title,author,content
    FROM blog
    WHERE id=${id}
</select>
```

仔细观察，**内联参数**的格式由“#{xxx}”变为了“${xxx}”。如果我们给参数“**id**”赋值为“**3**”，将 SQL 打印出来是这样的：

SELECT id,title,author,content FROM blog WHERE id = 3

（上面的对比示例是我自己添加的，为了与前面的示例形成鲜明的对比。）

```
<select id="orderBlog" resultType="Blog" parameterType="map">
    SELECT id,title,author,content
    FROM blog
    ORDER BY ${orderParam}
</select>
```

仔细观察，**内联参数**的格式由“#{xxx}”变为了“${xxx}”。如果我们给参数“**orderParam**”赋值为“**id**”，将 SQL 打印出来是这样的：

SELECT id,title,author,content FROM blog ORDER BY id

显然，这样是**无法阻止 SQL 注入**的。在 MyBatis 中，“${xxx}”这样格式的参数**会直接参与 SQL 编译**，从而**不能避免注入攻击**。但涉及到动态表名和列名时，只能使用“${xxx}”这样的参数格式。所以，这样的参数需要我们在代码中手工进行处理来防止注入。

【结论】在编写 MyBatis 的映射语句时，尽量采用“#{xxx}”这样的格式。若不得不使用“${xxx}”这样的参数，要手工地做好过滤工作，来防止 SQL 注入攻击。

[摘自] mybatis 的#{}和${}的区别以及 order by 注入问题

#{}：相当于 JDBC 中的 PreparedStatement

${}：是输出变量的值

简单说，#{}是经过**预编译的**，是**安全的**；${}是未经过预编译的，仅仅是取变量的值，是非安全的，存在 SQL 注入。

如果我们 order by 语句后用了${}，那么不做任何处理的时候是存在 SQL 注入危险的。你说怎么防止，那我只能悲惨的告诉你，你得手动处理过滤一下输入的内容。如判断一下输入的参数的长度是否正常（注入语句一般很长），**更精确的过滤**则可以查询一下输入的参数是否在预期的参数集合中。