1. #将传入的数据都当成一个字符串，会对自动传入的数据加一个双引号。如：order by #user\_id#，如果传入的值是111,那么解析成sql时的值为order by "111", 如果传入的值是id，则解析成的sql为order by "id".  
　　  
2. $将传入的数据直接显示生成在sql中。如：order by $user\_id$，如果传入的值是111,那么解析成sql时的值为order by user\_id,  如果传入的值是id，则解析成的sql为order by id.  
　　  
3. #方式能够很大程度防止sql注入。  
　　  
4.$方式无法防止Sql注入。  
  
5.$方式一般用于传入数据库对象，例如传入表名.  
　　  
6.一般能用#的就别用$.  
  
MyBatis排序时使用order by 动态参数时需要注意，用$而不是#  
  
字符串替换  
默认情况下，使用#{}格式的语法会导致MyBatis创建预处理语句属性并以它为背景设置安全的值（比如?）。这样做很安全，很迅速也是首选做法，有时你只是想直接在SQL语句中插入一个不改变的字符串。比如，像ORDER BY，你可以这样来使用：  
ORDER BY ${columnName}  
这里MyBatis不会修改或转义字符串。

重要：接受从用户输出的内容并提供给语句中不变的字符串，这样做是不安全的。这会导致潜在的SQL注入攻击，因此你不应该允许用户输入这些字段，或者通常自行转义并检查。

mybatis本身的说明：

String Substitution

By default, using the #{} syntax will cause MyBatis to generate PreparedStatement properties and set the values safely against the PreparedStatement parameters (e.g. ?). While this is safer, faster and almost always preferred, sometimes you just want to directly inject a string unmodified into the SQL Statement. For example, for ORDER BY, you might use something like this:

ORDER BY ${columnName}

Here MyBatis won't modify or escape the string.

NOTE It's not safe to accept input from a user and supply it to a statement unmodified in this way. This leads to potential SQL Injection attacks and therefore you should either disallow user input in these fields, or always perform your own escapes and checks.

从上文可以看出：

1. 使用#{}格式的语法在mybatis中使用Preparement语句来安全的设置值，执行sql类似下面的：

PreparedStatement ps = conn.prepareStatement(sql);

ps.setInt(1,id);

这样做的好处是：更安全，更迅速，通常也是首选做法。

而且#{}格式的sql能够预编译，能再内存中保存sql语法，不用重新组装sql语法。

2. 不过有时你只是想直接在 SQL 语句中插入一个不改变的字符串。比如，像 ORDER BY，你可以这样来使用：

ORDER BY ${columnName}

此时MyBatis 不会修改或转义字符串。

这种方式类似于：

Statement st = conn.createStatement();

ResultSet rs = st.executeQuery(sql);

这种方式的缺点是： 以这种方式接受从用户输出的内容并提供给语句中不变的字符串是不安全的，会导致潜在的 SQL 注入攻击，因此要么不允许用户输入这些字段，要么自行转义并检验。