php mysql注入_php连接mysql的三种方式和预处理下的sql注入

原创

游戏葡萄 于 2021-01-18 20:24:28 发布 57 收藏
文章标签： php mysql注入

0x00 前言

学习了一下堆叠注入和这三种连接方式预处理下的SQL注入问题。

0x01 基础知识

参考：

https://www.cnblogs.com/joshua317/articles/5989781.html

https://www.cnblogs.com/geaozhang/p/9891338.html

1、即时 SQL

一条 SQL 在 DB 接收到最终执行完毕返回，大致的过程如下：

1. 词法和语义解析；

2. 优化 SQL 语句，制定执行计划；

3. 执行并返回结果；

如上，一条 SQL 直接是走流程处理，一次编译，单次运行，此类普通语句被称作 Immediate Statements (即时 SQL)。

2、预处理 SQL

但是，绝大多数情况下，某需求某一条 SQL 语句可能会被反复调用执行，或者每次执行的时候只有个别的值不同(比如 select 的 where 子句值不同，update 的 set 子句值不同，insert 的 values 值不同)。如果每次都需要经过上面的词法语义解析、语句优化、制定执行计划等，则效率就明显不行了。

所谓预编译语句就是将此类 SQL 语句中的值用占位符替代，可以视为将 SQL 语句模板化或者说参数化，一般称这类语句叫Prepared Statements。

预编译语句的优势在于归纳为：一次编译、多次运行，省去了解析优化等过程；此外预编译语句能防止 SQL 注入。

3:PHP与MySQL的连接有三种API接口，分别是：PHP的MySQL扩展 、PHP的mysqli扩展 、PHP数据对象 (PDO)

	Mysqli	PDO	MySQL
引入的PHP版本	5.0	5.0	3.0之前
PHP5.x是否包含	是	是	是
服务端prepare语句的支持情况	是	是	否
客户端prepare语句的支持情况	否	是	否
存储过程支持情况	是	是	否
多语句执行支持情况	是	大多数	否

我们对比一下看看两者的区别，可以看到Mysqli和PDO是都是支持多语句执行的，这就为我要在后面写的注入埋下伏笔。

MySQL扩展

PHP的MySQL扩展是设计开发允许php应用与MySQL数据库交互的早期扩展。MySQL扩展提供了一个面向过程的接口，并且是针对MySQL4.1.3或者更早版本设计的。因此这个扩展虽然可以与MySQL4.1.3或更新的数据库服务端进行交互，但并不支持后期MySQL服务端提供的一些特性。由于太古老，又不安全，所以已被后来的mysqli完全取代；

mysqli扩展

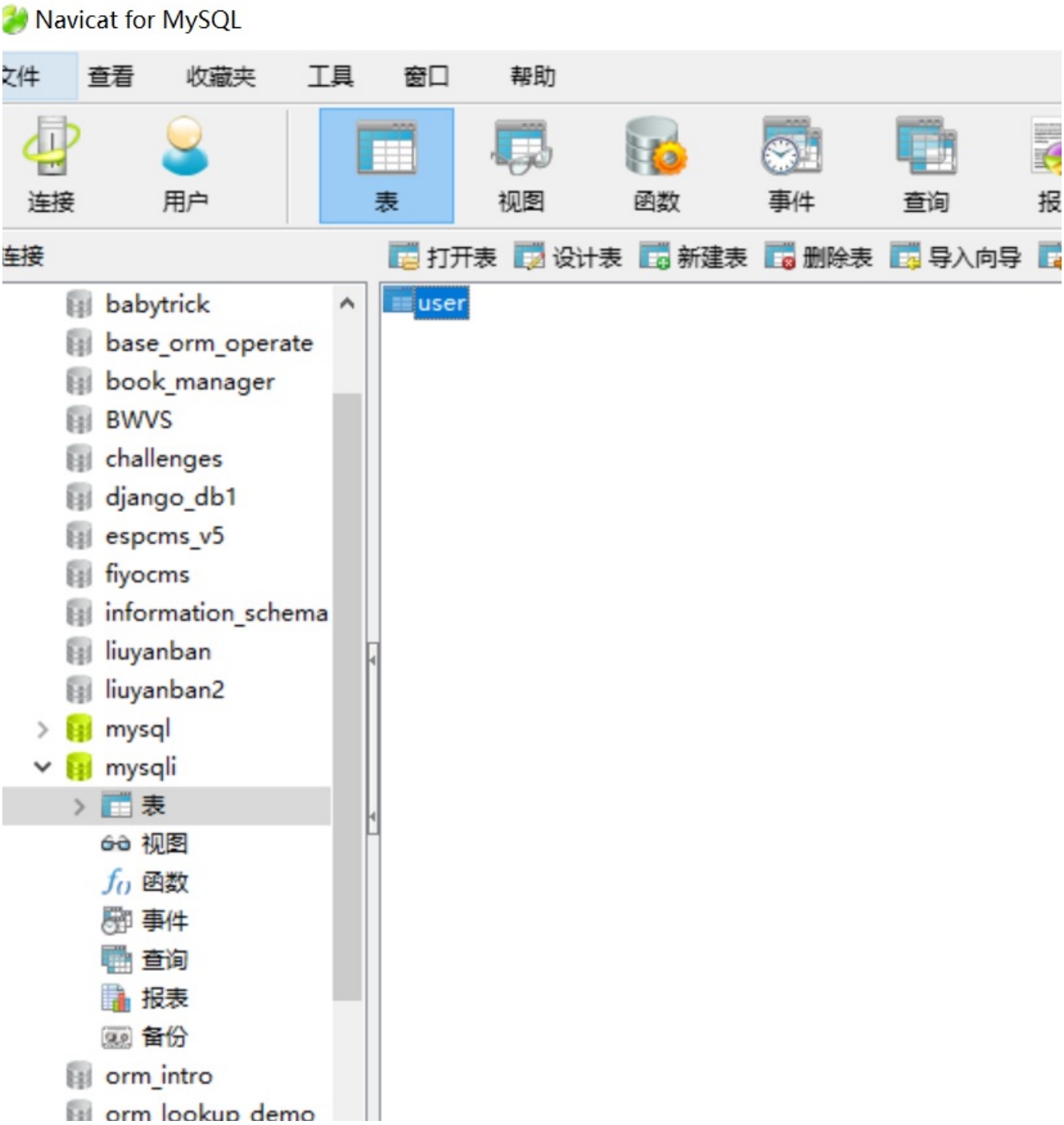
PHP的mysqli扩展，我们有时称之为MySQL增强扩展，可以用于使用 MySQL4.1.3或更新版本中新的高级特性。其特点为：面向对象接口 、prepared语句支持、多语句执行支持、事务支持 、增强的调试能力、嵌入式服务支持 、预处理方式完全解决了sql注入的问题。不过其也有缺点，就是只支持mysql数据库。如果你要是不操作其他的数据库，这无疑是最好的选择。

Mysqli通过multi_query()函数来进行多语句执行

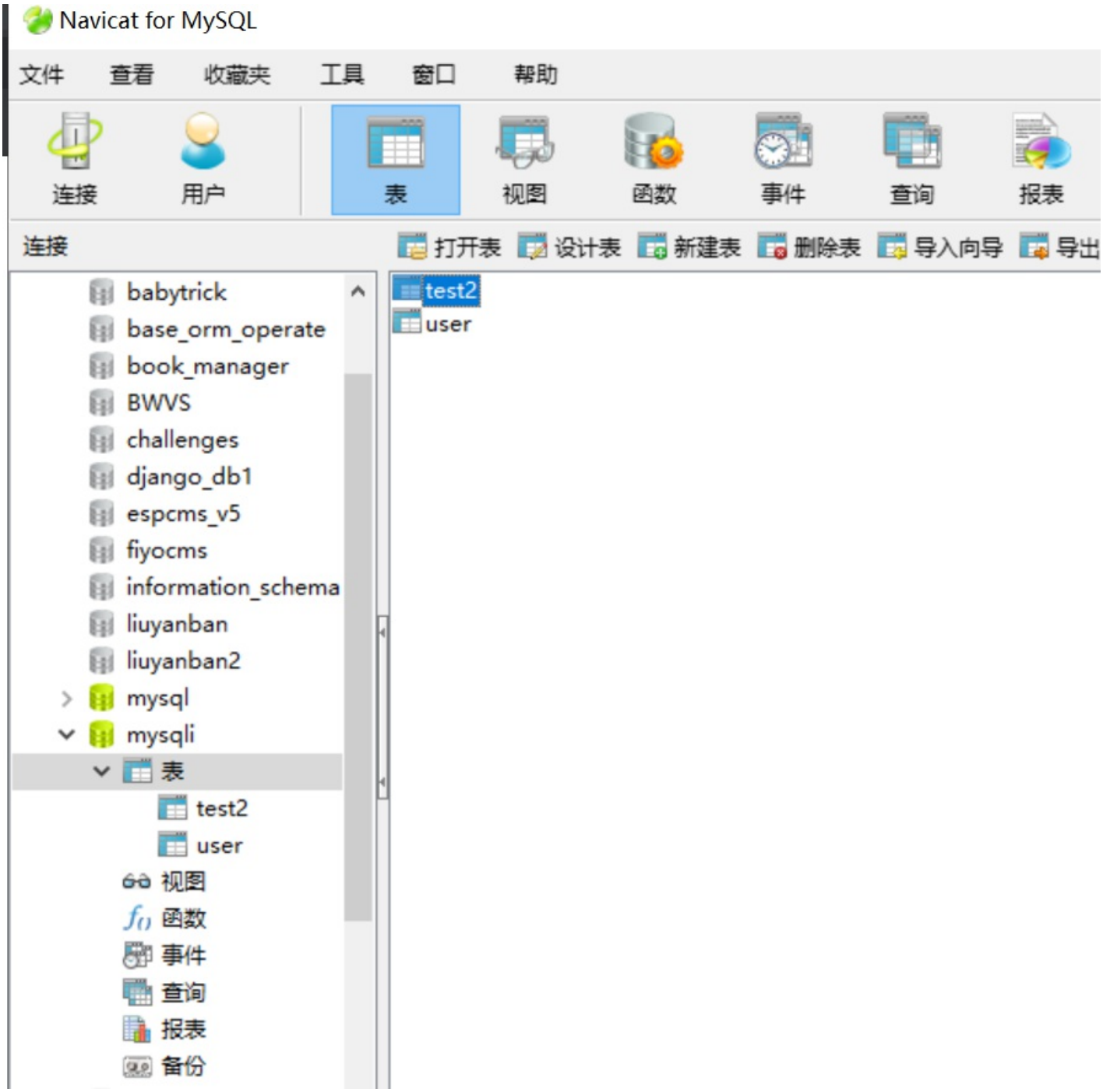
```
$host='127.0.0.1';

$dbName='mysqli';

$user='root';

$pass='root';

$mysqli = mysqli_connect($host,$user,$pass,$dbName);

if(mysqli_connect_errno())

{

echo mysqli_connect_error();

}

$sql = "select * from user where id=1;";
```

```
$sql .= "create table test2 like user";

$mysqli->multi_query($sql);

$data = $mysqli->store_result();

print_r($data->fetch_row());

mysqli_close($mysqli);
```

我们在本地建立一个user表，然后请求我们的php



发现数据库中成功创建了test2表，说明多语句成功执行

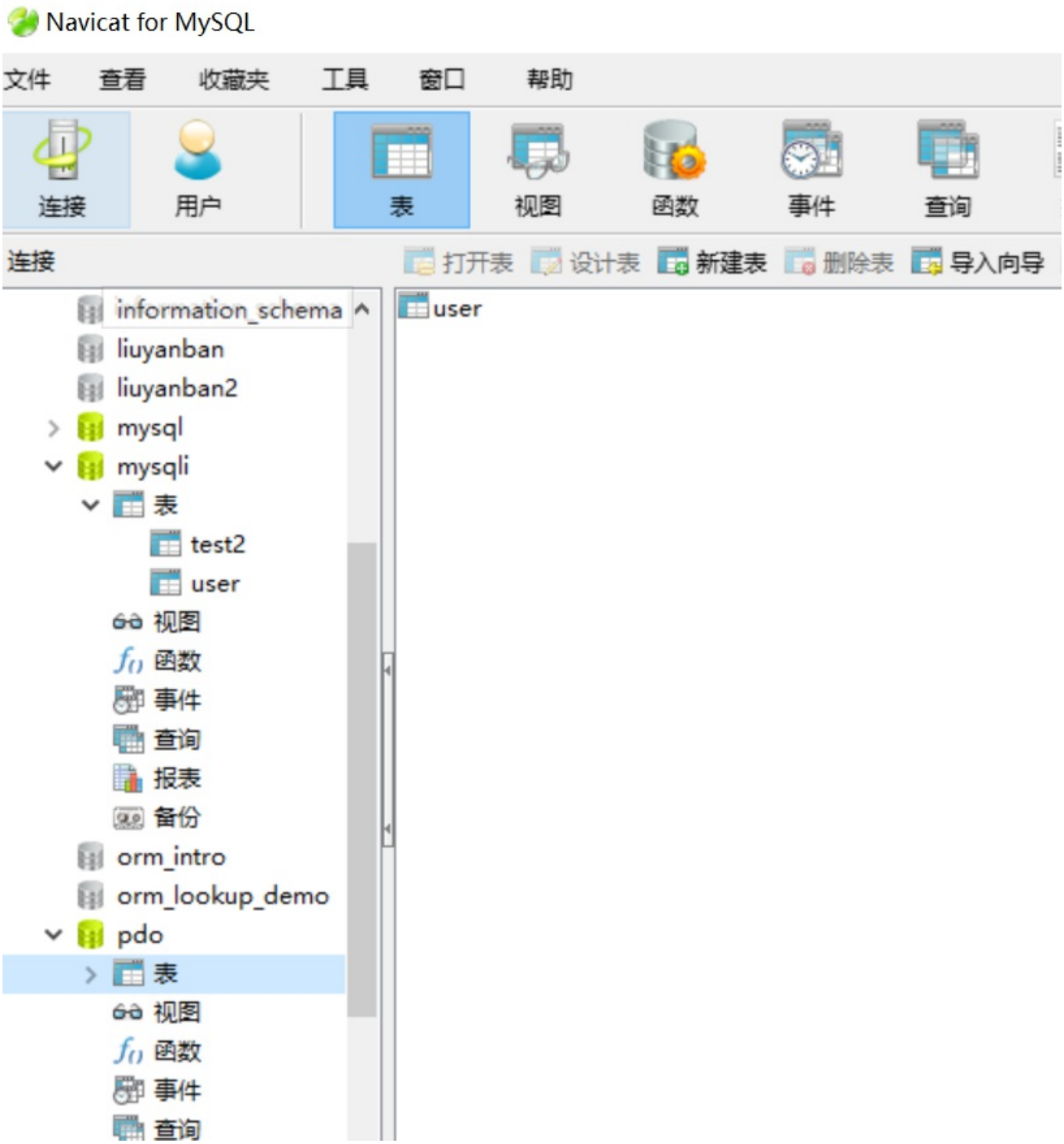


PDO

PDO是PHP Data Objects的缩写，是PHP应用中的一个数据库抽象层规范。PDO提供了一个统一的API接口可以使得你的PHP应用不去关心具体要连接的数据库服务器系统类型，也就是说，如果你使用PDO的API，可以在任何需要的时候无缝切换数据库服务器，比如从Oracle 到MySQL，仅仅需要修改很少的PHP代码。其功能类似于JDBC、ODBC、DBI之类接口。同样，其也解决了sql注入问题，有很好的安全性。不过他也有缺点，某些多语句执行查询不支持(不过该情况很少)。

同样的，我们使用PDO中的query()函数同数据库交互

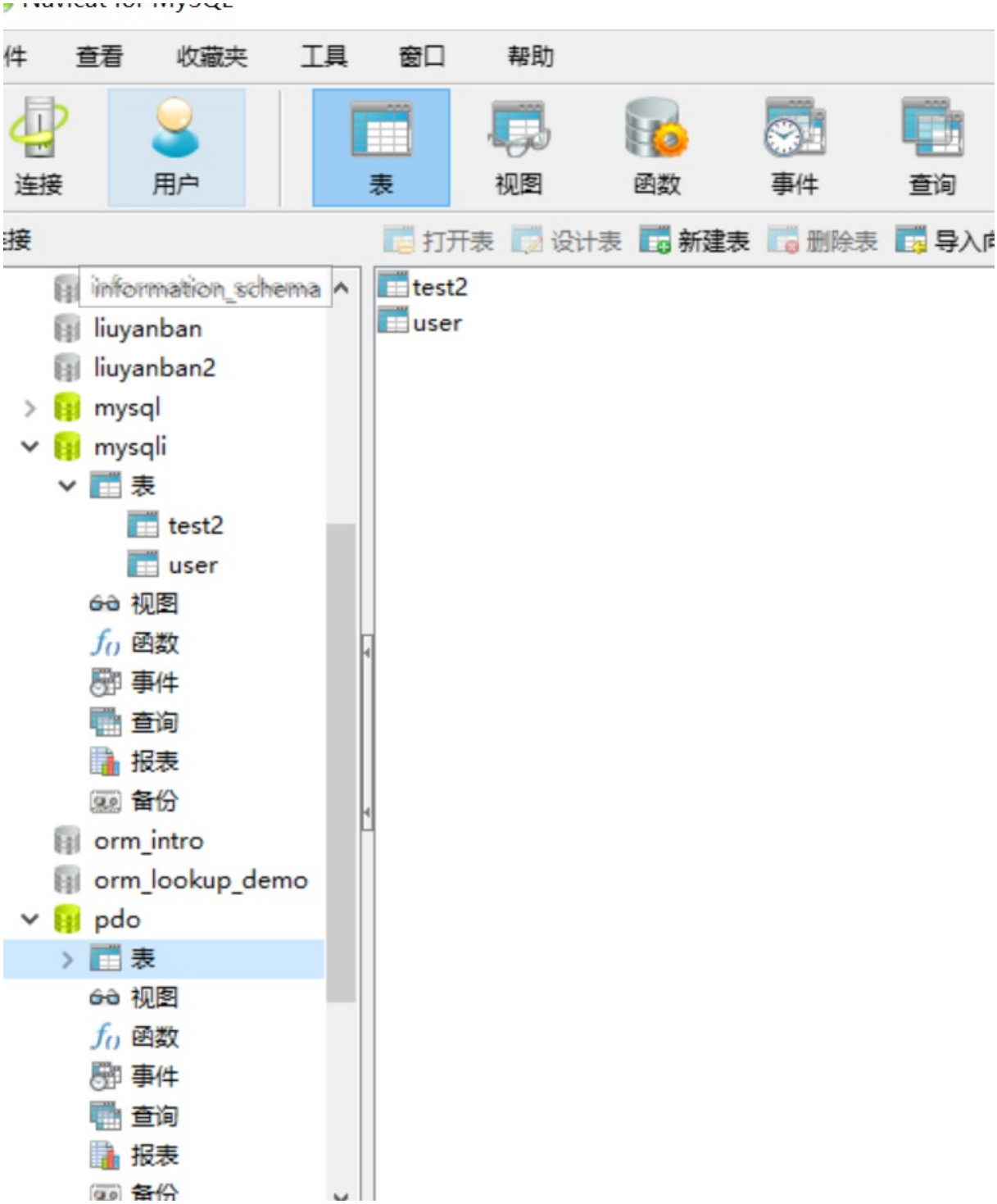
先新建一个数据库




```
$dbms='mysql';
$host='127.0.0.1';
$dbName='pdo';
$user='root';
$pass='root';
$dsn="$dbms:host=$host;dbname=$dbName";
try {
$pdo = new PDO($dsn, $user, $pass);
} catch (PDOException $e) {
echo $e;
```

```
}

$sql = "select * from user where id=1;";

$sql .= "create table test2 like user";

$stmt = $pdo->query($sql);

while($row=$stmt->fetch(PDO::FETCH_ASSOC))

{

var_dump($row);

echo "
";

}
```

运行我们的php文件后发现数据库中成功创建了test2表，说明多语句成功执行



再来看一个：

PDO默认支持多语句查询，如果php版本小于5.5.21或者创建PDO实例时未设置PDO::MYSQL_ATTR_MULTI_STATEMENTS为false时可能会造成堆叠注入

```
$dbms='mysql';

$host='127.0.0.1';

$dbName='pdo';

$user='root';

$pass='root';

$dsn="$dbms:host=$host;dbname=$dbName";
```

```
try {
$pdo = new PDO($dsn, $user, $pass);
} catch (PDOException $e) {
echo $e;
}
$id = $_GET['id'];
$sql = "SELECT * from user where id =".$id;
$stmt = $pdo->query($sql);
while($row=$stmt->fetch(PDO::FETCH_ASSOC))
{
var_dump($row);
echo "
";
}
```

如果想禁止多语句执行，可在创建PDO实例时将PDO::MYSQL_ATTR_MULTI_STATEMENTS设置为false

```
new PDO($dsn, $user, $pass, array( PDO::MYSQL_ATTR_MULTI_STATEMENTS => false))
```

mysql预处理和问题

MySQL数据库支持预处理，预处理或者说是可传参的语句用来高效的执行重复的语句。

MySQL官方将prepare、execute、deallocate统称为PREPARE STATEMENT

预制语句的SQL语法基于三个SQL语句：

```
// 定义预处理语句
PREPARE stmt_name FROM preparable_stmt;
// 执行预处理语句
EXECUTE stmt_name [USING @var_name [, @var_name] ...];
//删除(释放)定义
{DEALLOCATE | DROP} PREPARE stmt_name;
```

我们先来看一个正常的预处理：

```
mysql> SET @a=0x73656c656374202a2066726f6d2075736572;
Query OK, 0 rows affected (0.00 sec)

mysql> prepare execsql from @a;
Query OK, 0 rows affected (0.00 sec)
Statement prepared

mysql> execute execsql;
+----+----------+----------------------------------+---------------------+-------------------
-----------------------------+------------+
| id | username | password                         | email               | pic
                            | join_date  |
+----+----------+----------------------------------+---------------------+-------------------
-----------------------------+------------+
|  3 |          | c22a40fc1112b3de6ae058169b9f2b20 | 6666                | 11212
                            | 2020-03-02 |
|  4 | admin    | admin                            | 6666                | 11212
```

解释：

set用于设置变量名和值

prepare用于预备一个语句，并赋予名称，以后可以引用该语句

execute执行语句

deallocate prepare用来释放掉预处理的语句

这里我用16进制来表示user表赋值给变量a

然后预备一个名叫execsql的语句。

然后执行。

而如果想要执行这么多语句，我们必须要能够多语句执行，因此这个想法也就造就了堆叠注入。

堆叠注入的使用条件十分有限，其可能受到API或者数据库引擎，又或者权限的限制只有当调用数据库函数支持执行多条sql语句时才能够使用，利用mysqli_multi_query()函数就支持多条sql语句同时执行，但实际情况中，如PHP为了防止sql注入机制，往往使用调用数据库的函数是mysqli_ query()函数，其只能执行一条语句，分号后面的内容将不会被执行，所以可以说堆叠注入的使用条件十分有限，一旦能够被使用，将可能对网站造成十分大的威胁。

想了解的可以去看一个ctf题目

[强网杯 2019]随便注，以前也写过

PDO预处理和问题

PDO默认支持多语句查询，如果php版本小于5.5.21或者创建PDO实例时未设置
PDO::MYSQL_ATTR_MULTI_STATEMENTS为false时可能会造成堆叠注入

PDO分为模拟预处理和非模拟预处理。

模拟预处理是防止某些数据库不支持预处理而设置的，在初始化PDO驱动时，可以设置一项参数，PDO::ATTR_EMULATE_PREPARES，作用是打开模拟预处理(true)或者关闭(false),默认为true。PDO内部会模拟参数绑定的过程，SQL语句是在最后execute()的时候才发送给数据库执行。

非模拟预处理则是通过数据库服务器来进行预处理动作，主要分为两步：第一步是prepare阶段，发送SQL语句模板到数据库服务器；第二步通过execute()函数发送占位符参数给数据库服务器进行执行。

PDO与安全问题相关的主要的设置有下面三个:

PDO::ATTR_EMULATE_PREPARES

PDO::ATTR_ERRMODE

PDO::MYSQL_ATTR_MULTI_STATEMENTS

在分别开启的情况下分别与模拟预编译、报错和多句执行有关。

PDO默认是允许多句执行和模拟预编译的,上面也提过了，在参数可控的情况下，会导致堆叠注入。

问题很大的模拟预编译

在模拟预编译的情况下，PDO对于SQL注入的防范(PDO::queto())，无非就是将数字型的注入转变为字符型的注入，又用类似mysql_real_escape_string()的方法将单引号、双引号、反斜杠等字符进行了转义。

这种防范方法在GBK编码的情况下便可用宽字节进行绕过。

非GBK编码的情况下，若存在二次注入的情况，是否能利用呢？

二次注入是由于对添加进数据库中的数据没有再次处理和转义而导致的，而预编译对每次查询都进行转义，则不存在二次注入的情况。

上述安全隐患，是由于未正确设置PDO造成的，在PDO的默认设置中，PDO::ATTR_EMULATE_PREPARES和PDO::MYSQL_ATTR_MULTI_STATEMENTS都是true，意味着模拟预编译和多句执行是默认开启的。

而在非模拟预编译的情况下，若语句中没有可控参数,则不可以注入。

结论

非模拟预编译的情况下，若语句中没有可控参数

如果不是GBK编码，如上面所说，也不存在二次注入的情况，故可以避免SQL注入漏洞。

PDO中的的Prepare Statement方法

PDO的原理，与Mysql中prepare语句是一样的。上面我已经演示过了。

这样设置不用担心没有合理地设置PDO，或是用了GBK编码等情况。

Prepare Statement在SQL注入中的利用

Prepare语句最大的特点就是它可以将16进制串转为语句字符串并执行。如果我们发现了一个存在堆叠注入的场景，但过滤非常严格，便可以使用prepare语句进行绕过。

利用方式同上

总结

合理、安全地使用gbk编码。即使采用PDO预编译的方式，如若配置不当，依然可造成宽字节注入

使用PDO时，一定要将模拟预编译设为false

可采用使用Prepare Statement手动预编译，杜绝SQL注入

创建PDO实例时将PDO::MYSQL_ATTR_MULTI_STATEMENTS设置为false，禁止多语句查询。

参考