

# 数据库与身份认证



黑马程序员™  
www.itheima.com

传智播客旗下  
高端IT教育品牌



# 目录 Contents

- ◆ 数据库的基本概念
- ◆ 安装并配置 MySQL
- ◆ MySQL的基本使用
- ◆ 在项目中操作 MySQL
- ◆ 前后端的身份认证



# 1. 数据库的基本概念

## 1.1 什么是数据库

数据库 (database) 是用来组织、存储和管理数据的仓库。

当今世界是一个充满着数据的互联网世界，充斥着大量的数据。数据的来源有很多，比如出行记录、消费记录、浏览的网页、发送的消息等等。除了文本类型的数据，图像、音乐、声音都是数据。

为了方便管理互联网世界中的数据，就有了数据库管理系统的概念（简称：数据库）。用户可以对数据库中的数据进行新增、查询、更新、删除等操作。





# 1. 数据库的基本概念

## 1.2 常见的数据库及分类

市面上的数据库有很多种，最常见的数据库有如下几个：

- MySQL 数据库（目前使用最广泛、流行度最高的开源免费数据库：Community + Enterprise）
- Oracle 数据库（收费）
- SQL Server 数据库（收费）
- Mongodb 数据库（Community + Enterprise）

其中，MySQL、Oracle、SQL Server 属于传统型数据库（又叫做：关系型数据库 或 SQL 数据库），这三者的设计理念相同，用法比较类似。

而 Mongodb 属于新型数据库（又叫做：非关系型数据库 或 NoSQL 数据库），它在一定程度上弥补了传统型数据库的缺陷。



# 1. 数据库的基本概念

## 1.3 传统型数据库的数据组织结构

数据的组织结构：指的就是数据以什么样的结构进行存储。



传统型数据库的数据组织结构，与 Excel 中数据的数据的组织结构比较类似。

因此，我们可以对比着 Excel 来了解和学习传统型数据库的数据组织结构。



# 1. 数据库的基本概念

## 1.3 传统型数据库的数据组织结构

### 1. Excel 的数据组织结构

每个 Excel 中，数据的组织结构分别为**工作簿**、**工作表**、**数据行**、**列**这 4 大部分组成。

	A	B	C	D	E	F
1	编号	用户名	密码	年龄	性别	用户状态
2	1	zs	123456	22	男	1
3	2	ls	000000	25	女	0
4	3	admin	abc123	28	男	0
5						
6						
7						
8						
9						
10						

- ① 整个 Excel 叫做**工作簿**
- ② users 和 books 是**工作表**
- ③ users 工作表中有 3 行数据
- ④ 每行数据由 6 列信息组成
- ⑤ 每列信息都有对应的**数据类型**

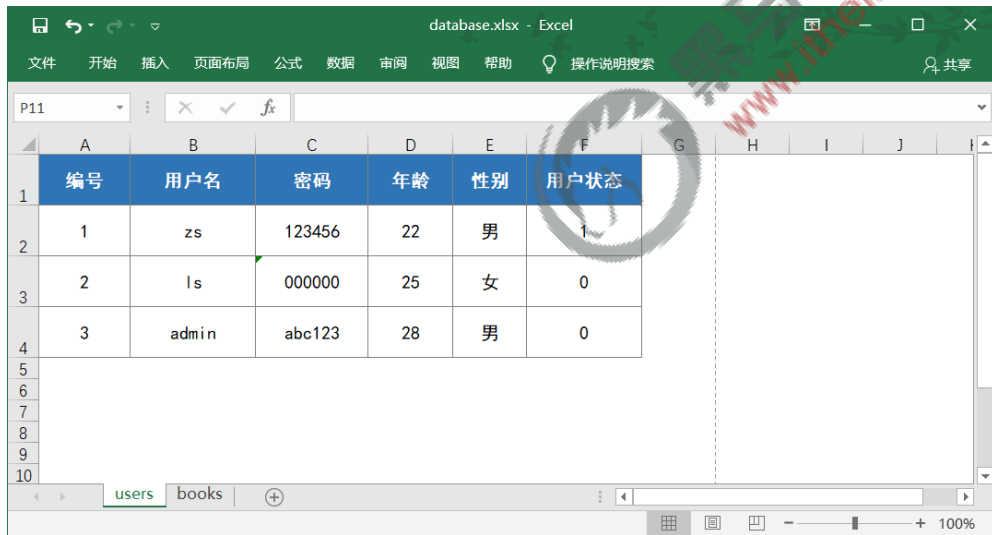


# 1. 数据库的基本概念

## 1.3 传统型数据库的数据组织结构

### 2. 传统型数据库的数据组织结构

在传统型数据库中，数据的组织结构分为数据库(database)、数据表(table)、数据行(row)、字段(field)这 4 大部分组成。



	编号	用户名	密码	年龄	性别	用户状态
1	1	zs	123456	22	男	1
2	2	ls	000000	25	女	0
3	3	admin	abc123	28	男	0

- ① 数据库类似于 Excel 的工作簿
- ② 数据表类似于 Excel 的工作表
- ③ 数据行类似于 Excel 的每一行数据
- ④ 字段类似于 Excel 的列
- ⑤ 每个字段都有对应的数据类型



# 1. 数据库的基本概念

## 1.3 传统型数据库的数据组织结构

### 3. 实际开发中库、表、行、字段的关系

- ① 在实际项目开发中，一般情况下，每个项目都对应**独立的数据库**。
- ② 不同的数据，要存储到数据库的不同表中，例如：用户数据存储到 users 表中，图书数据存储到 books 表中。
- ③ 每个表中具体存储哪些信息，由字段来决定，例如：我们可以为 users 表设计 id、username、password 这 3 个字段。
- ④ 表中的行，代表每一条具体的数据。



# 目录 Contents

- ◆ 数据库的基本概念
- ◆ 安装并配置 MySQL
- ◆ MySQL的基本使用
- ◆ 在项目中操作 MySQL
- ◆ 前后端的身份认证

## ■ 2. 安装并配置 MySQL

### 2.1 了解需要安装哪些MySQL相关的软件

对于开发人员来说，只需要安装 [MySQL Server](#) 和 [MySQL Workbench](#) 这两个软件，就能满足开发的需要了。

- MySQL Server: 专门用来提供数据存储和服务的软件。
- MySQL Workbench: 可视化的 MySQL 管理工具，通过它，可以方便的操作存储在 MySQL Server 中的数据。



## 2. 安装并配置 MySQL

### 2.2 MySQL 在 Mac 环境下的安装

在 Mac 环境下安装 MySQL 的过程比 Windows 环境下的步骤简单很多：

- ① 先运行 **mysql-8.0.19-macos10.15-x86\_64.dmg** 这个安装包，将 MySQL Server 安装到 Mac 系统
- ② 再运行 **mysql-workbench-community-8.0.19-macos-x86\_64.dmg** 这个安装包，将可视化的 MySQL Workbench 工具安装到 Mac 系统

具体的安装教程，可以参考 [素材 -> MySQL for Mac](#) -> [安装教程 - Mac系统安装MySQL](#) -> [README.md](#)

## ■ 2. 安装并配置 MySQL

### 2.3 MySQL 在 Windows 环境下的安装

在 Windows 环境下安装 MySQL，只需要运行 **mysql-installer-community-8.0.19.0.msi** 这个安装包，就能一次性将 MySQL Server 和 MySQL Workbench 安装到自己的电脑上。

具体的安装教程，可以参考 [素材](#) -> [MySQL for Windows](#) -> [安装教程 - Windows系统安装MySQL](#) -> [README.md](#)



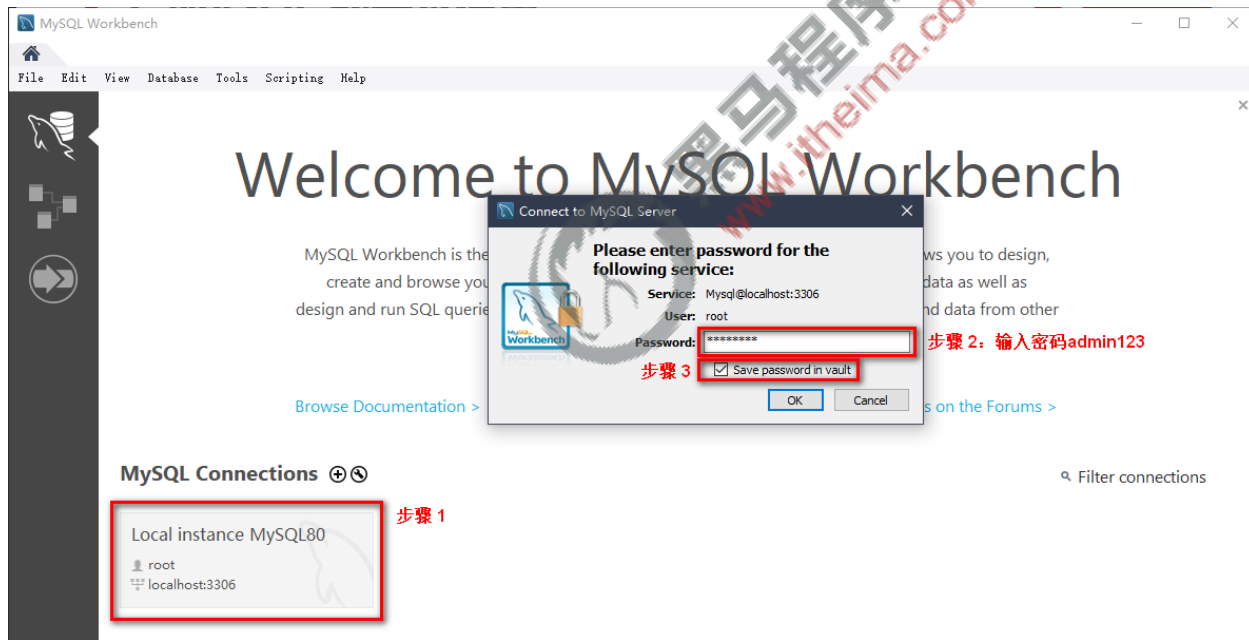
# 目录 Contents

- ◆ 数据库的基本概念
- ◆ 安装并配置 MySQL
- ◆ MySQL的基本使用
- ◆ 在项目中操作 MySQL
- ◆ 前后端的身份认证

# 3. MySQL 的基本使用

## 3.1 使用 MySQL Workbench 管理数据库

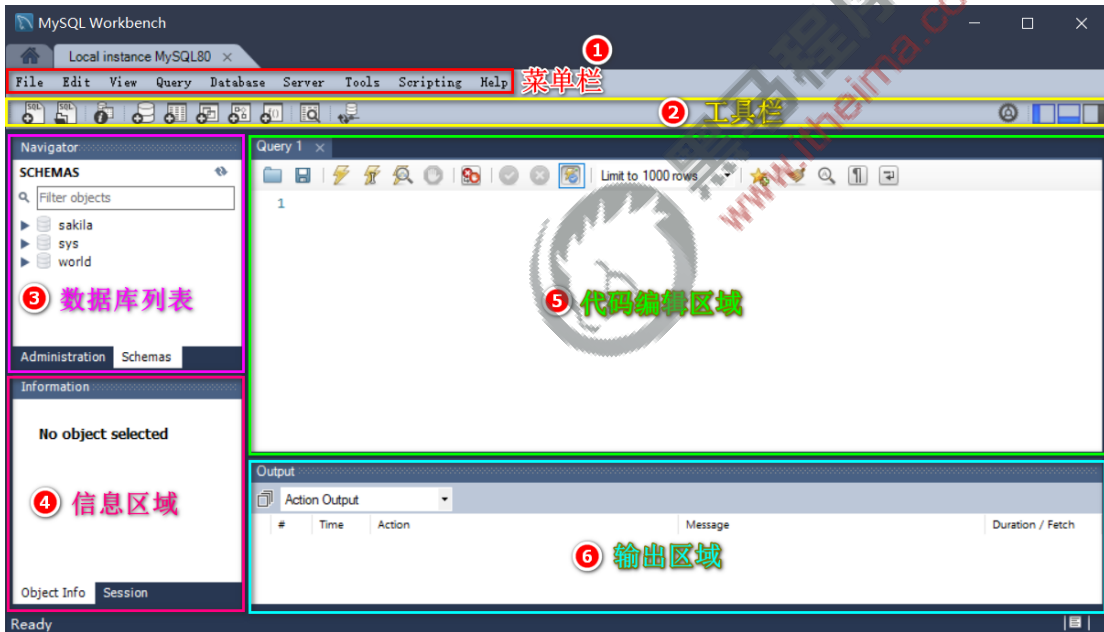
### 1. 连接数据库



# 3. MySQL 的基本使用

## 3.1 使用 MySQL Workbench 管理数据库

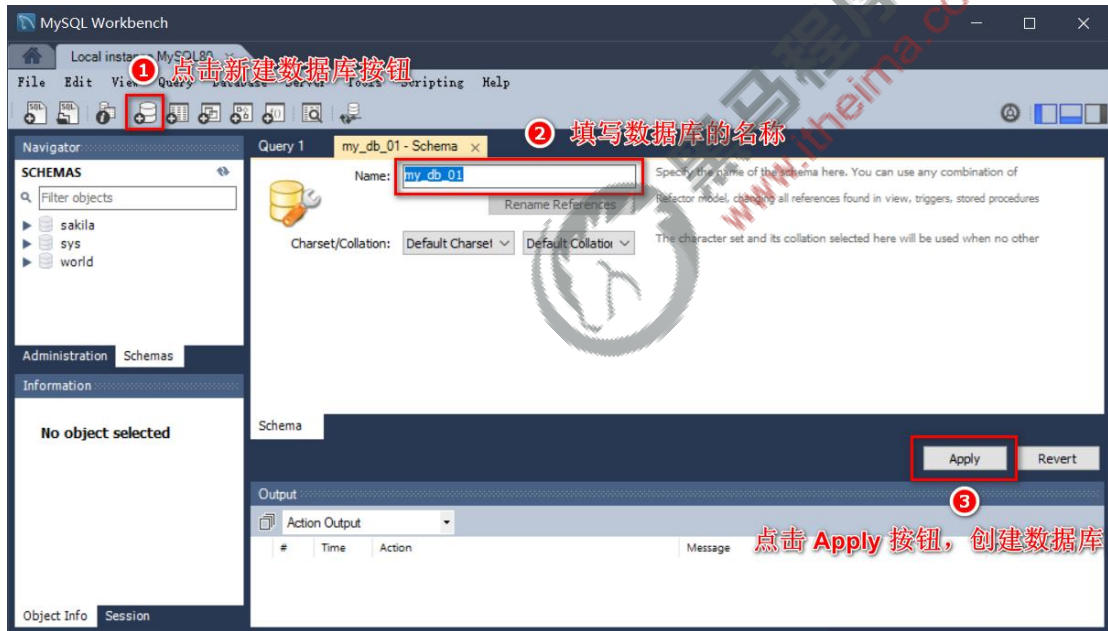
### 2. 了解主界面的组成部分



## 3. MySQL 的基本使用

### 3.1 使用 MySQL Workbench 管理数据库

#### 3. 创建数据库

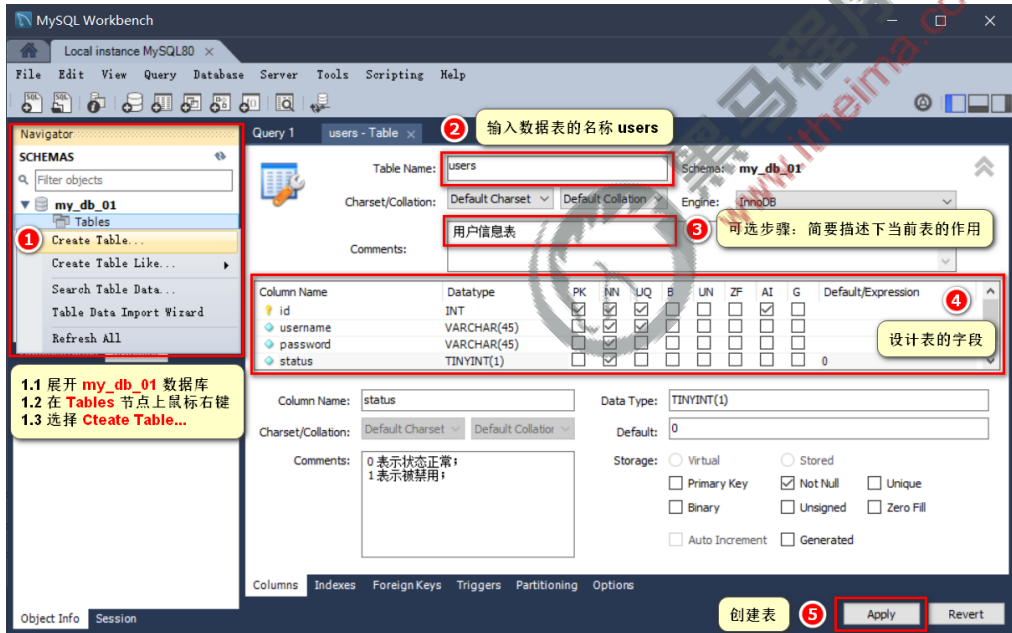




# 3. MySQL 的基本使用

## 3.1 使用 MySQL Workbench 管理数据库

### 4. 创建数据表



DataType 数据类型：

- ① **int** 整数
- ② **varchar(len)** 字符串
- ③ **tinyint(1)** 布尔值

字段的特殊标识：

- ① **PK** (Primary Key) 主键、唯一标识
- ② **NN** (Not Null) 值不允许为空
- ③ **UQ** (Unique) 值唯一
- ④ **AI** (Auto Increment) 值自动增长

# 3. MySQL 的基本使用

## 3.1 使用 MySQL Workbench 管理数据库

### 5. 向表中写入数据

1.1 在 **users** 表上鼠标右键  
1.2 选择 **Select Rows - Limit1000**

2.1 向 **users** 表中, 输入 **zs** 和 **ls** 两条数据, 他们的密码分别是 **123456** 和 **abc123**  
2.2 其中, 字段 **id** 不需要手动输入, 因为它是自增字段  
2.3 其中, 字段 **status** 也不需要手动输入, 因为它默认值为 **0**, 表示用户状态正常

保存对 **users** 表的修改

3. **Apply**

# 3. MySQL 的基本使用

## 3.2 使用 SQL 管理数据库

### 1. 什么是 SQL

SQL (英文全称: Structured Query Language) 是结构化查询语言, 专门用来访问和处理数据库的编程语言。能够让我们以编程的形式, 操作数据库里面的数据。

三个关键点:

- ① SQL 是一门数据库编程语言
- ② 使用 SQL 语言编写出来的代码, 叫做 SQL 语句
- ③ SQL 语言只能在关系型数据库中使用 (例如 MySQL、Oracle、SQL Server)。非关系型数据库 (例如 MongoDB) 不支持 SQL 语言

# ■ 3. MySQL 的基本使用

## 3.2 使用 SQL 管理数据库

### 2. SQL 能做什么

- ① 从数据库中查询数据
- ② 向数据库中插入新的数据
- ③ 更新数据库中的数据
- ④ 从数据库删除数据
- ⑤ 可以创建新数据库
- ⑥ 可在数据库中创建新表
- ⑦ 可在数据库中创建存储过程、视图
- ⑧ etc...



## ■ 3. MySQL 的基本使用

### 3.2 使用 SQL 管理数据库

#### 3. SQL 的学习目标

重点掌握如何使用 SQL 从数据表中：

查询数据 (select) 、 插入数据 (insert into) 、 更新数据 (update) 、 删除数据 (delete)

额外需要掌握的 4 种 SQL 语法：

where 条件、 and 和 or 运算符、 order by 排序、 count(\*) 函数

## 3. MySQL 的基本使用

### 3.3 SQL 的 SELECT 语句

#### 1. 语法

SELECT 语句用于从表中查询数据。执行的结果被存储在一个结果表中（称为结果集）。语法格式如下：

```
1 -- 这是注释
2 -- 从 FROM 指定的【表中】，查询出【所有的】数据。 * 表示【所有列】
3 SELECT * FROM 表名称
4
5 -- 从 FROM 指定的【表中】，查询出指定 列名称（字段） 的数据。
6 SELECT 列名称 FROM 表名称
```

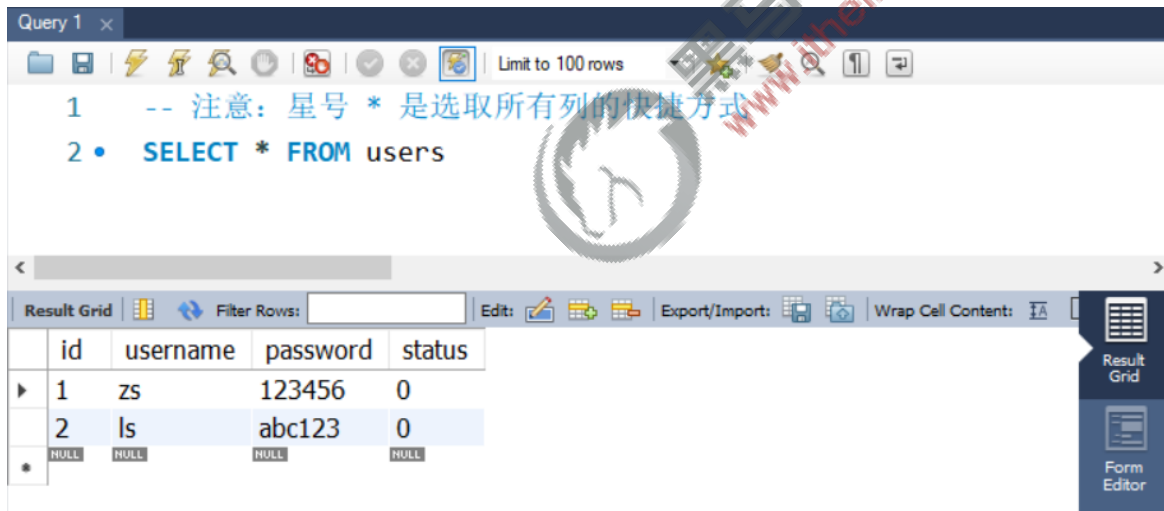
注意：SQL 语句中的关键字对大小写不敏感。SELECT 等效于 select，FROM 等效于 from。

## 3. MySQL 的基本使用

### 3.3 SQL 的 SELECT 语句

#### 2. SELECT \* 示例

我们希望从 users 表中选取所有的列，可以使用符号 \* 取代列的名称，示例如下：



The screenshot shows a MySQL query editor window titled "Query 1". The query text is:

```
1 -- 注意：星号 * 是选取所有列的快捷方式
2 • SELECT * FROM users
```

Below the query editor, the "Result Grid" is displayed, showing the results of the query. The grid has four columns: id, username, password, and status. The results are as follows:

	id	username	password	status
▶	1	zs	123456	0
	2	ls	abc123	0
*	NULL	NULL	NULL	NULL

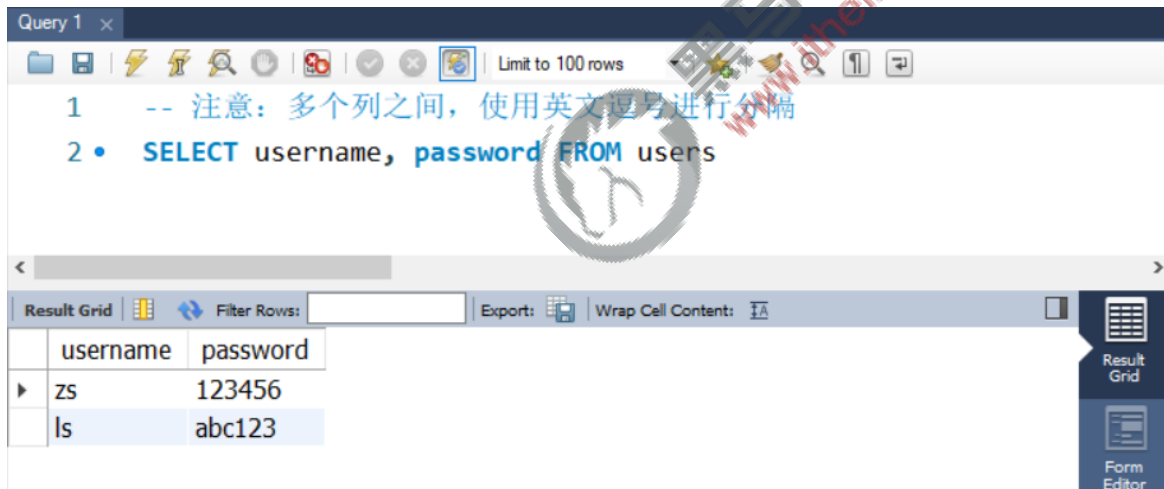
The interface also includes a toolbar with various icons and a "Limit to 100 rows" option. The "Result Grid" and "Form Editor" tabs are visible on the right side.

## 3. MySQL 的基本使用

### 3.3 SQL 的 SELECT 语句

#### 3. SELECT 列名称 示例

如需获取名为 "username" 和 "password" 的列的内容（从名为 "users" 的数据库表），请使用下面的 SELECT 语句：



The screenshot shows a MySQL query editor window titled "Query 1". The query text is:

```
1  -- 注意：多个列之间，使用英文逗号进行分隔
2  •  SELECT username, password FROM users
```

Below the query editor, the "Result Grid" is displayed, showing the results of the query. The grid has two columns: "username" and "password". The results are:

username	password
zs	123456
ls	abc123

The interface also includes a toolbar with various icons, a "Limit to 100 rows" dropdown, and a "Filter Rows" input field. The "Result Grid" and "Form Editor" buttons are visible on the right side of the interface.



## 3. MySQL 的基本使用

### 3.4 SQL 的 INSERT INTO 语句

#### 1. 语法

INSERT INTO 语句用于向数据表中插入新的数据行，语法格式如下：

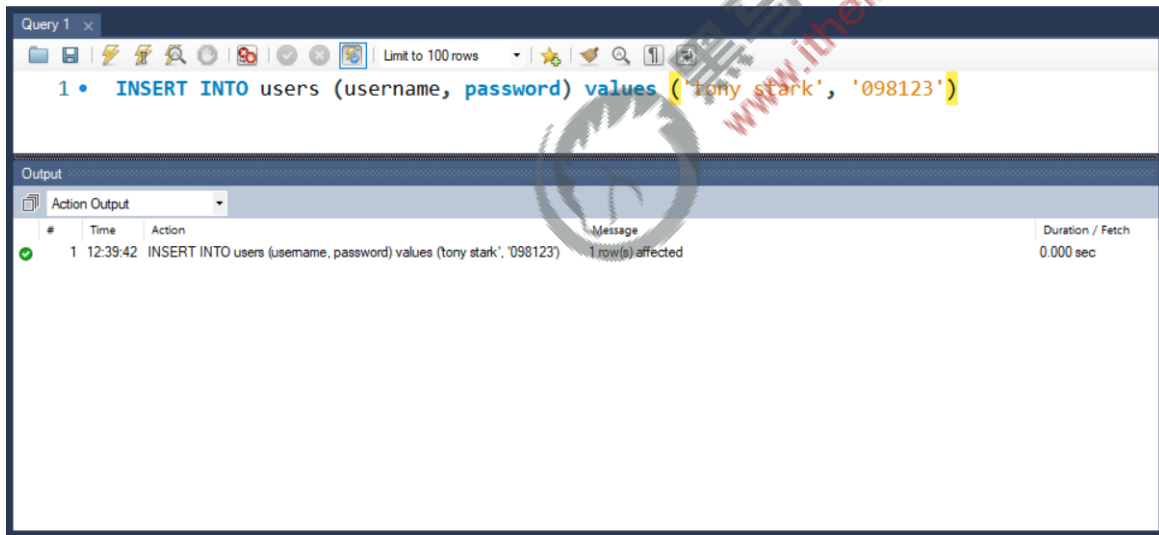
```
1 -- 语法解读：向指定的表中，插入如下几列数据，列的值通过 values 指定
2 -- 注意： 列和值要一一对应，多个列和多个值之间，使用英文的逗号分隔
3 INSERT INTO table_name (列1, 列2,...) VALUES (值1, 值2,...)
```

## 3. MySQL 的基本使用

### 3.4 SQL 的 INSERT INTO 语句

#### 2. INSERT INTO 示例

向 users 表中，插入一条 username 为 tony stark, password 为 098123 的用户数据，示例如下：



## 3. MySQL 的基本使用

### 3.5 SQL 的 UPDATE 语句

#### 1. 语法

Update 语句用于修改表中的数据。语法格式如下：

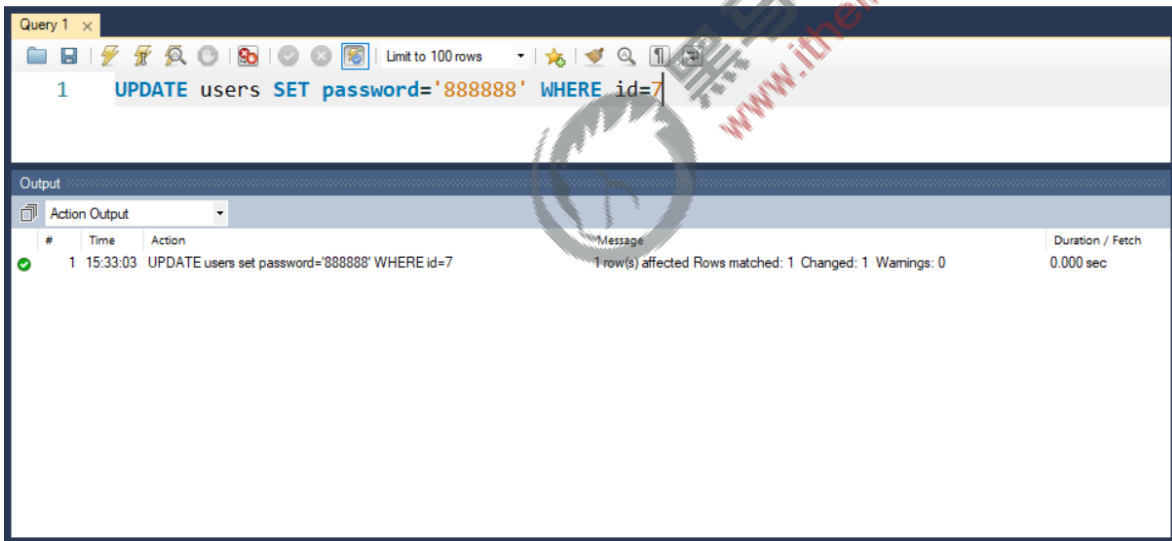
```
1 -- 语法解读：
2 -- 1. 用 UPDATE 指定要更新哪个表中的数据
3 -- 2. 用 SET 指定列对应的新值
4 -- 3. 用 WHERE 指定更新的条件
5 UPDATE 表名称 SET 列名称 = 新值 WHERE 列名称 = 某值
```

## 3. MySQL 的基本使用

### 3.5 SQL 的 UPDATE 语句

#### 2. UPDATE 示例 - 更新某一行中的一个列

把 users 表中 id 为 7 的用户密码，更新为 888888。示例如下：

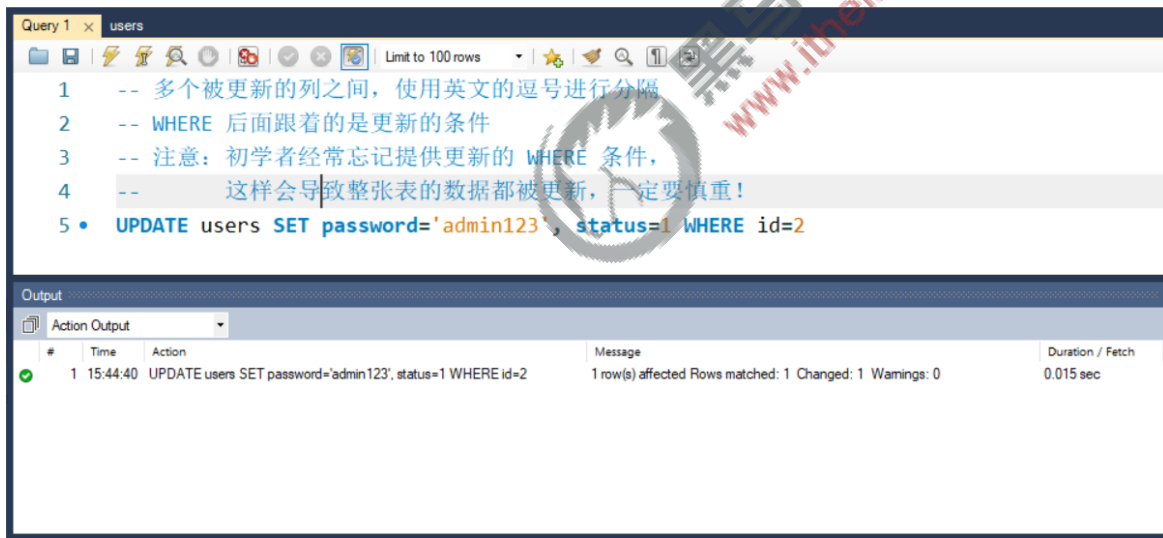


## 3. MySQL 的基本使用

### 3.5 SQL 的 UPDATE 语句

#### 3. UPDATE 示例 - 更新某一行中的若干列

把 users 表中 id 为 2 的用户密码和用户状态，分别更新为 admin123 和 1。示例如下：



The screenshot shows a MySQL IDE window titled 'Query 1 x users'. The query editor contains the following SQL statement:

```
1  -- 多个被更新的列之间，使用英文的逗号进行分隔
2  -- WHERE 后面跟着的是更新的条件
3  -- 注意：初学者经常忘记提供更新的 WHERE 条件，
4  -- 这样会导致整张表的数据都被更新，一定要慎重！
5 • UPDATE users SET password='admin123', status=1 WHERE id=2
```

Below the query editor is the 'Output' panel, which shows the 'Action Output' for the executed query:

#	Time	Action	Message	Duration / Fetch
1	15:44:40	UPDATE users SET password='admin123', status=1 WHERE id=2	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.015 sec

## 3. MySQL 的基本使用

### 3.6 SQL 的 DELETE 语句

#### 1. 语法

**DELETE** 语句用于删除表中的行。语法格式如下：

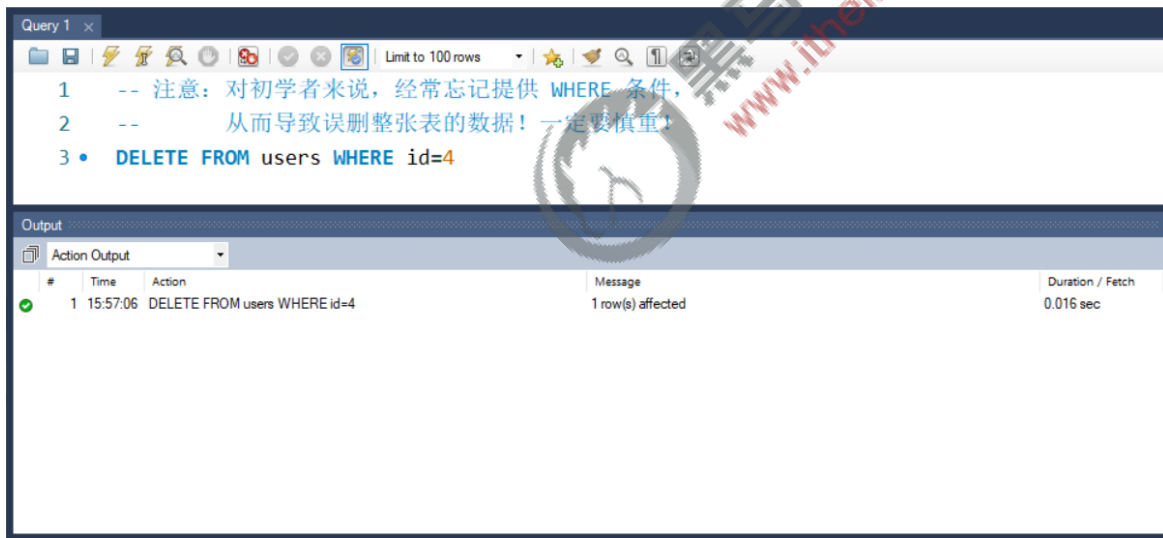
```
1 -- 语法解读:  
2 -- 从指定的表中, 根据 WHERE 条件, 删除对应的数据行  
3 DELETE FROM 表名称 WHERE 列名称 = 值
```

## 3. MySQL 的基本使用

### 3.6 SQL 的 DELETE 语句

#### 2. DELETE 示例

从 users 表中，删除 id 为 4 的用户，示例如下：



The screenshot shows a MySQL query editor window titled "Query 1". The query text is as follows:

```
1  -- 注意：对初学者来说，经常忘记提供 WHERE 条件，
2  --      从而导致误删整张表的数据！一定要慎重！
3  •  DELETE FROM users WHERE id=4
```

Below the query editor is the "Output" panel, which shows the execution results in a table format:

#	Time	Action	Message	Duration / Fetch
✓ 1	15:57:06	DELETE FROM users WHERE id=4	1 row(s) affected	0.016 sec

## 3. MySQL 的基本使用

### 3.7 SQL 的 WHERE 子句

#### 1. 语法

WHERE 子句用于限定选择的标准。在 SELECT、UPDATE、DELETE 语句中，皆可使用 WHERE 子句来限定选择的标准。

```
1 -- 查询语句中的 WHERE 条件
2 SELECT 列名称 FROM 表名称 WHERE 列 运算符 值
3 -- 更新语句中的 WHERE 条件
4 UPDATE 表名称 SET 列=新值 WHERE 列 运算符 值
5 -- 删除语句中的 WHERE 条件
6 DELETE FROM 表名称 WHERE 列 运算符 值
```



## 3. MySQL 的基本使用

### 3.7 SQL 的 WHERE 子句

#### 2. 可在 WHERE 子句中使用的运算符

下面的运算符可在 WHERE 子句中使用，用来限定选择的标准：

操作符	描述
=	等于
<>	不等于
>	大于
<	小于
>=	大于等于
<=	小于等于
BETWEEN	在某个范围内
LIKE	搜索某种模式

注意：在某些版本的 SQL 中，操作符 <> 可以写为 !=

## 3. MySQL 的基本使用

### 3.7 SQL 的 WHERE 子句

#### 3. WHERE 子句示例

可以通过 WHERE 子句来限定 SELECT 的查询条件:

```
1 -- 查询 status 为 1 的所有用户
2 SELECT * FROM users WHERE status=1
3 -- 查询 id 大于 2 的所有用户
4 SELECT * FROM users WHERE id>2
5 -- 查询 username 不等于 admin 的所有用户
6 SELECT * FROM users WHERE username<>'admin'
```

## 3. MySQL 的基本使用

### 3.8 SQL 的 AND 和 OR 运算符

#### 1. 语法

AND 和 OR 可在 WHERE 子语句中把两个或多个条件结合起来。

AND 表示必须同时满足多个条件，相当于 JavaScript 中的 && 运算符，例如 `if (a !== 10 && a !== 20)`

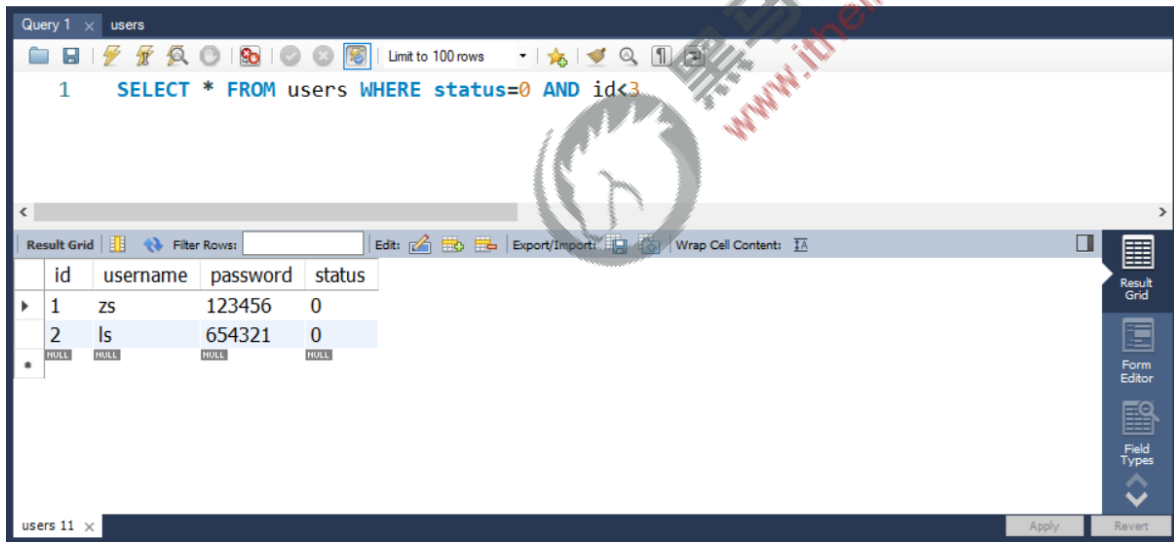
OR 表示只要满足任意一个条件即可，相当于 JavaScript 中的 || 运算符，例如 `if(a !== 10 || a !== 20)`

## 3. MySQL 的基本使用

### 3.8 SQL 的 AND 和 OR 运算符

#### 2. AND 运算符示例

使用 AND 来显示所有 **status** 为 0, 并且 **id** 小于 3 的用户:



The screenshot shows a MySQL query editor window titled "Query 1 x users". The query entered is: `1 SELECT * FROM users WHERE status=0 AND id<3`. Below the query, the results are displayed in a table with columns: id, username, password, and status. The results show two rows: (1, zs, 123456, 0) and (2, ls, 654321, 0). The interface includes a toolbar with icons for saving, undo, redo, and other database operations. A watermark for "黑马程序员" and "www.itheima.com" is visible across the image.

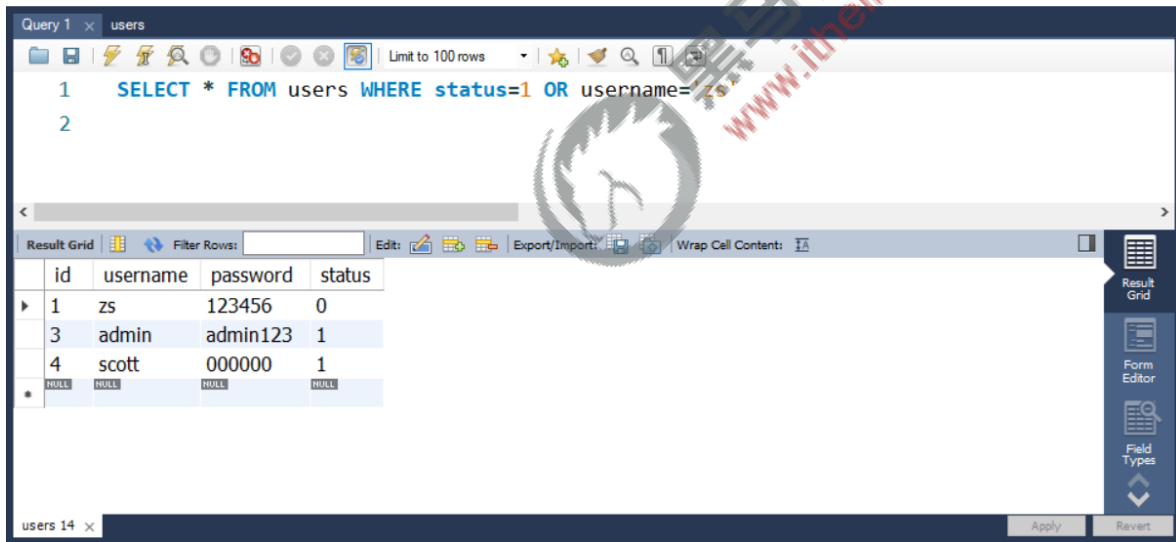
	id	username	password	status
1	zs	123456	0	
2	ls	654321	0	

# 3. MySQL 的基本使用

## 3.8 SQL 的 AND 和 OR 运算符

### 2. OR 运算符示例

使用 OR 来显示所有 status 为 1, 或者 username 为 zs 的用户:



The screenshot shows a MySQL query editor window. The query entered is:

```
1 SELECT * FROM users WHERE status=1 OR username='zs'
2
```

The results are displayed in a table with the following data:

	id	username	password	status
1	zs	123456	0	
3	admin	admin123	1	
4	scott	000000	1	
*	NULL	NULL	NULL	NULL

The interface includes a toolbar with various icons, a 'Limit to 100 rows' dropdown, and a sidebar with options like 'Result Grid', 'Form Editor', and 'Field Types'. The status bar at the bottom shows 'users 14' and 'Apply Revert' buttons.

## ■ 3. MySQL 的基本使用

### 3.9 SQL 的 ORDER BY 子句

#### 1. 语法

ORDER BY 语句用于根据指定的列对结果集进行排序。

ORDER BY 语句默认按照升序对记录进行排序。

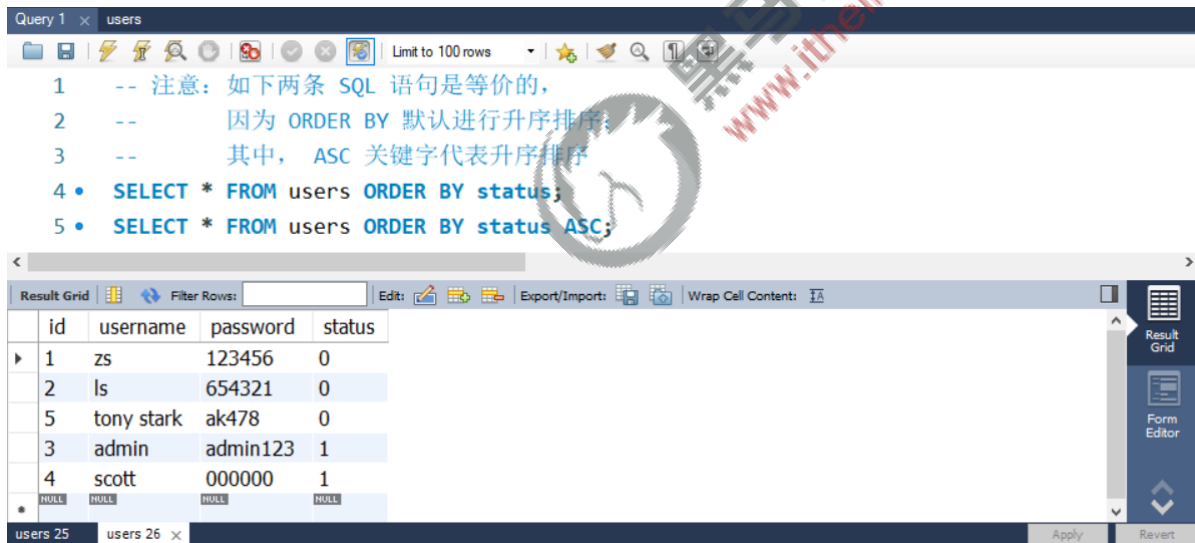
如果您希望按照降序对记录进行排序，可以使用 DESC 关键字。

## 3. MySQL 的基本使用

### 3.9 SQL 的 ORDER BY 子句

#### 2. ORDER BY 子句 - 升序排序

对 users 表中的数据，按照 status 字段进行升序排序，示例如下：



The screenshot shows a MySQL query editor window titled "Query 1 x users". The SQL query entered is:

```
1 -- 注意：如下两条 SQL 语句是等价的，  
2 -- 因为 ORDER BY 默认进行升序排序；  
3 -- 其中，ASC 关键字代表升序排序  
4 • SELECT * FROM users ORDER BY status;  
5 • SELECT * FROM users ORDER BY status ASC;
```

Below the query, the "Result Grid" shows the results of the query, sorted by status. The results are as follows:

	id	username	password	status
1	zs	123456	0	
2	ls	654321	0	
5	tony stark	ak478	0	
3	admin	admin123	1	
4	scott	000000	1	
*	NULL	NULL	NULL	NULL

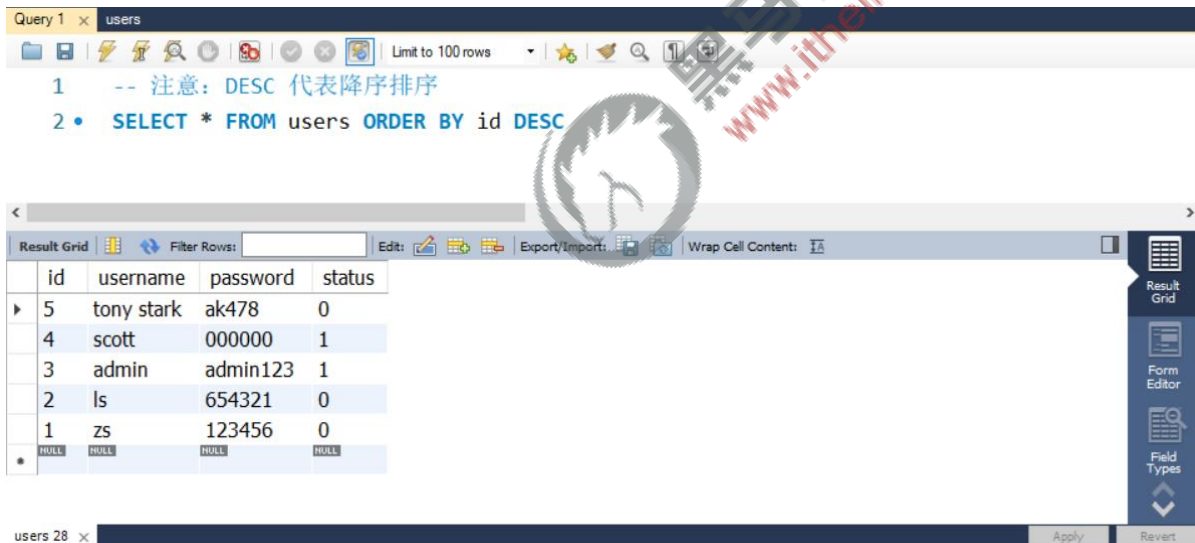
The bottom status bar shows "users 25" and "users 26 x".

## 3. MySQL 的基本使用

### 3.9 SQL 的 ORDER BY 子句

#### 3. ORDER BY 子句 – 降序排序

对 users 表中的数据，按照 id 字段进行降序排序，示例如下：



The screenshot shows a MySQL query editor window titled 'Query 1 x users'. The SQL query entered is:

```
1 -- 注意: DESC 代表降序排序
2 • SELECT * FROM users ORDER BY id DESC
```

The results are displayed in a table with the following columns: id, username, password, and status. The data is sorted by id in descending order.

	id	username	password	status
▶	5	tony stark	ak478	0
	4	scott	000000	1
	3	admin	admin123	1
	2	ls	654321	0
	1	zs	123456	0
•	NULL	NULL	NULL	NULL

The interface includes a toolbar with various icons for editing and viewing the results, and a sidebar on the right with options like 'Result Grid', 'Form Editor', and 'Field Types'.

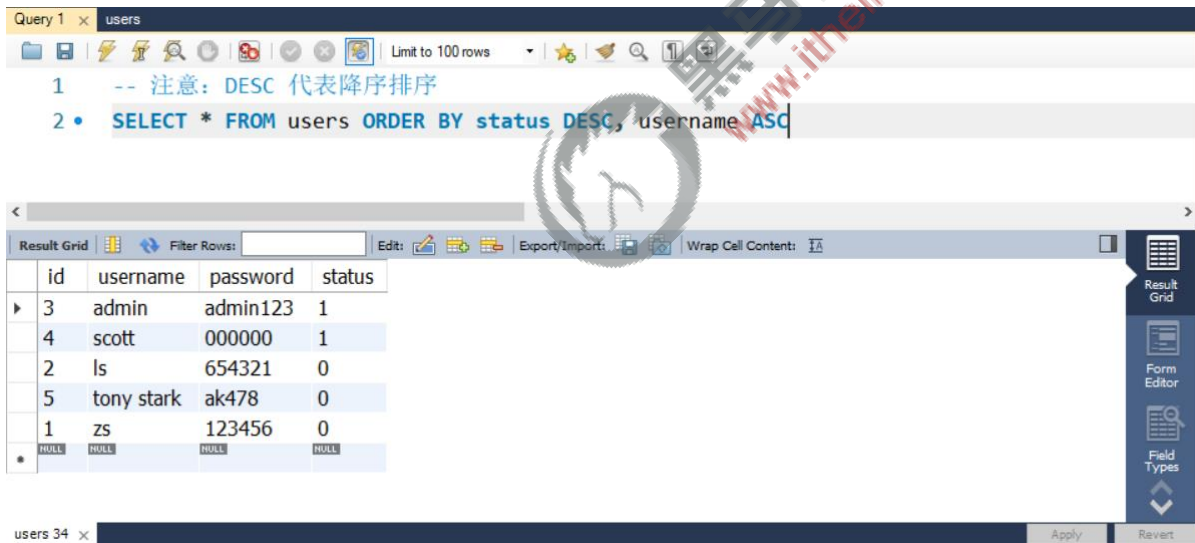


## 3. MySQL 的基本使用

### 3.9 SQL 的 ORDER BY 子句

#### 4. ORDER BY 子句 – 多重排序

对 users 表中的数据，先按照 status 字段进行降序排序，再按照 username 的字母顺序，进行升序排序，示例如下：



The screenshot shows a MySQL query editor window titled "Query 1 x users". The SQL query entered is:

```
1 -- 注意: DESC 代表降序排序
2 • SELECT * FROM users ORDER BY status DESC, username ASC
```

The results are displayed in a table with the following columns: id, username, password, and status. The data is sorted by status in descending order, and then by username in ascending order for rows with the same status.

	id	username	password	status
▶	3	admin	admin123	1
	4	scott	000000	1
	2	ls	654321	0
	5	tony stark	ak478	0
	1	zs	123456	0
*	NULL	NULL	NULL	NULL

The bottom of the window shows "users 34 x" and buttons for "Apply" and "Revert".

## 3. MySQL 的基本使用

### 3.10 SQL 的 COUNT(\*) 函数

#### 1. 语法

COUNT(\*) 函数用于返回查询结果的总数据条数，语法格式如下：

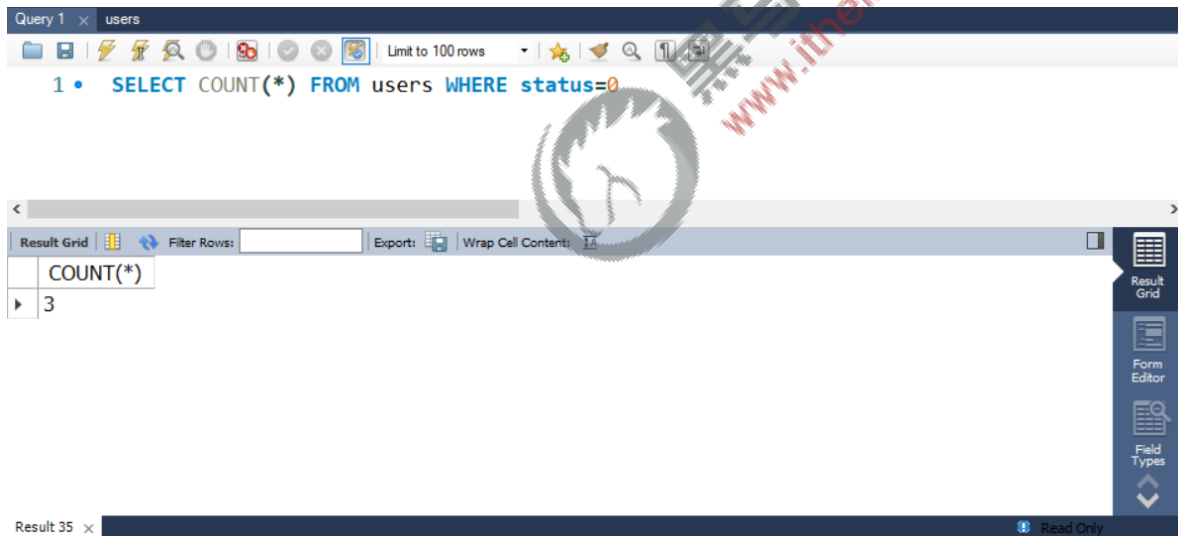
```
1 SELECT COUNT(*) FROM 表名称
```

## 3. MySQL 的基本使用

### 3.10 SQL 的 COUNT(\*) 函数

#### 2. COUNT(\*) 示例

查询 users 表中 status 为 0 的总数据条数:



The screenshot shows a MySQL query editor window titled "Query 1" with the following SQL query:

```
1 • SELECT COUNT(*) FROM users WHERE status=0
```

The query results are displayed in a table with the following structure:

COUNT(*)
3

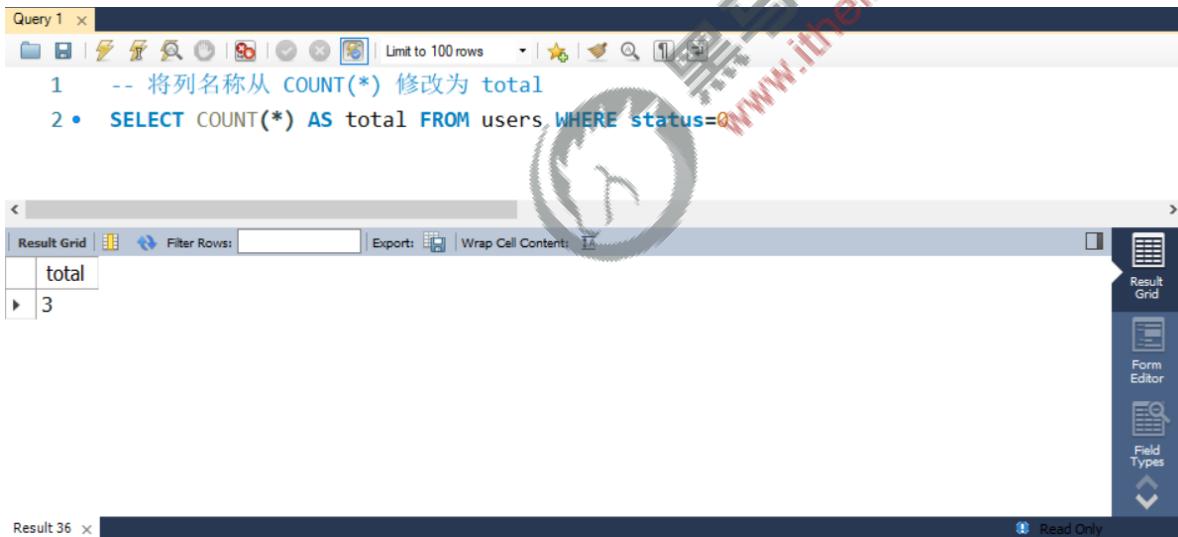
The interface includes a toolbar with various icons, a "Limit to 100 rows" dropdown, and a "Result Grid" button. The status bar at the bottom indicates "Result 35" and "Read Only".

## 3. MySQL 的基本使用

### 3.10 SQL 的 COUNT(\*) 函数

#### 2. 使用 AS 为列设置别名

如果希望给查询出来的列名称设置别名，可以使用 AS 关键字，示例如下：



# 目录 Contents

- ◆ 数据库的基本概念
- ◆ 安装并配置 MySQL
- ◆ MySQL的基本使用
- ◆ 在项目中操作 MySQL
- ◆ 前后端的身份认证

## 4. 在项目中操作 MySQL

### 4.1 在项目中操作数据库的步骤

- ① 安装操作 MySQL 数据库的第三方模块 (mysql)
- ② 通过 mysql 模块连接到 MySQL 数据库
- ③ 通过 mysql 模块执行 SQL 语句



## 4. 在项目中操作 MySQL

### 4.2 安装与配置 mysql 模块

#### 1. 安装 mysql 模块

mysql 模块是托管于 npm 上的**第三方模块**。它提供了在 Node.js 项目中**连接**和**操作** MySQL 数据库的能力。

想要在项目中使用它，需要先运行如下命令，将 mysql 安装为项目的依赖包：

```
1 npm install mysql
```

## 4. 在项目中操作 MySQL

### 4.2 安装与配置 mysql 模块

#### 2. 配置 mysql 模块

在使用 mysql 模块操作 MySQL 数据库之前，必须先对 mysql 模块进行必要的配置，主要的配置步骤如下：

```
1 // 1. 导入 mysql 模块
2 const mysql = require('mysql')
3 // 2. 建立与 MySQL 数据库的连接
4 const db = mysql.createPool({
5   host: '127.0.0.1',    // 数据库的 IP 地址
6   user: 'root',         // 登录数据库的账号
7   password: 'admin123', // 登录数据库的密码
8   database: 'my_db_01' // 指定要操作哪个数据库
9 })
```



## 4. 在项目中操作 MySQL

### 4.2 安装与配置 mysql 模块

#### 3. 测试 mysql 模块能否正常工作

调用 `db.query()` 函数，指定要执行的 SQL 语句，通过回调函数拿到执行的结果：

```
1 // 检测 mysql 模块能否正常工作
2 db.query('SELECT 1', (err, results) => {
3   if (err) return console.log(err.message)
4   // 只要能打印出 [ RowDataPacket { '1': 1 } ] 的结果, 就证明数据库连接正常
5   console.log(results)
6 })
```

## 4. 在项目中操作 MySQL

### 4.3 使用 mysql 模块操作 MySQL 数据库

#### 1. 查询数据

查询 users 表中所有的数据：

```
1 // 查询 users 表中所有的用户数据
2 db.query('SELECT * FROM users', (err, results) => {
3   // 查询失败
4   if (err) return console.log(err.message)
5   // 查询成功
6   console.log(results)
7 })
```

## 4. 在项目中操作 MySQL

### 4.3 使用 mysql 模块操作 MySQL 数据库

#### 2. 插入数据

向 users 表中新增数据，其中 username 为 Spider-Man，password 为 pcc321。示例代码如下：

```
1 // 1. 要插入到 users 表中的数据对象
2 const user = { username: 'Spider-Man', password: 'pcc321' }
3 // 2. 待执行的 SQL 语句，其中英文的 ? 表示占位符
4 const sqlStr = 'INSERT INTO users (username, password) VALUES (?, ?)'
5 // 3. 使用数组的形式，依次为 ? 占位符指定具体的值
6 db.query(sqlStr, [user.username, user.password], (err, results) => {
7   if (err) return console.log(err.message) // 失败
8   if(results.affectedRows === 1) { console.log('插入数据成功') } // 成功
9 })
```

## 4. 在项目中操作 MySQL

### 4.3 使用 mysql 模块操作 MySQL 数据库

#### 3. 插入数据的便捷方式

向表中新增数据时，如果数据对象的每个属性和数据表的字段一一对应，则可以通过如下方式快速插入数据：

```
1 // 1. 要插入到 users 表中的数据对象
2 const user = { username: 'Spider-Man2', password: 'pcc4321' }
3 // 2. 待执行的 SQL 语句，其中英文的 ? 表示占位符
4 const sqlStr = 'INSERT INTO users SET ?'
5 // 3. 直接将数据对象当作占位符的值
6 db.query(sqlStr, user, (err, results) => {
7   if (err) return console.log(err.message) // 失败
8   if(results.affectedRows === 1) { console.log('插入数据成功') } // 成功
9 })
```

## 4. 在项目中操作 MySQL

### 4.3 使用 mysql 模块操作 MySQL 数据库

#### 4. 更新数据

可以通过如下方式，更新表中的数据：

```
1 // 1. 要更新的数据对象
2 const user = { id: 7, username: 'aaa', password: '000' }
3 // 2. 要执行的 SQL 语句
4 const sqlStr = 'UPDATE users SET username=?, password=? WHERE id=?'
5 // 3. 调用 db.query() 执行 SQL 语句的同时，使用数组依次为占位符指定具体的值
6 db.query(sqlStr, [user.username, user.password, user.id], (err, results) => {
7   if (err) return console.log(err.message) // 失败
8   if (results.affectedRows === 1) { console.log('更新数据成功! ') } // 成功
9 })
```

## 4. 在项目中操作 MySQL

### 4.3 使用 mysql 模块操作 MySQL 数据库

#### 5. 更新数据的便捷方式

更新表数据时，如果数据对象的每个属性和数据表的字段一一对应，则可以通过如下方式快速更新表数据：

```
1 // 1. 要更新的数据对象
2 const user = { id: 7, username: 'aaaa', password: '0000' }
3 // 2. 要执行的 SQL 语句
4 const sqlStr = 'UPDATE users SET ? WHERE id=?'
5 // 3. 调用 db.query() 执行 SQL 语句的同时，使用数组依次为占位符指定具体的值
6 db.query(sqlStr, [user, user.id], (err, results) => {
7   if (err) return console.log(err.message) // 失败
8   if (results.affectedRows === 1) { console.log('更新数据成功! ') } // 成功
9 })
```

## 4. 在项目中操作 MySQL

### 4.3 使用 mysql 模块操作 MySQL 数据库

#### 6. 删除数据

在删除数据时，推荐根据 id 这样的唯一标识，来删除对应的数据。示例如下：

```
1 // 1. 要执行的 SQL 语句
2 const sqlStr = 'DELETE FROM users WHERE id=?'
3 // 2. 调用 db.query() 执行 SQL 语句的同时，为占位符指定具体的值
4 // 注意：如果 SQL 语句中有多个占位符，则必须使用数组为每个占位符指定具体的值
5 //      如果 SQL 语句中只有一个占位符，则可以省略数组
6 db.query(sqlStr, 7, (err, results) => {
7   if (err) return console.log(err.message) // 失败
8   if (results.affectedRows === 1) { console.log('删除数据成功! ') } // 成功
9 })
```

## 4. 在项目中操作 MySQL

### 4.3 使用 mysql 模块操作 MySQL 数据库

#### 7. 标记删除

使用 DELETE 语句，会把真正的把数据从表中删除掉。为了保险起见，**推荐使用标记删除**的形式，来**模拟删除的动作**。

所谓的标记删除，就是在表中设置类似于 **status** 这样的**状态字段**，来**标记**当前这条数据是否被删除。

当用户执行了删除的动作时，我们并没有执行 DELETE 语句把数据删除掉，而是执行了 UPDATE 语句，将这条数据对应的 status 字段标记为删除即可。

```
1 // 标记删除: 使用 UPDATE 语句替代 DELETE 语句; 只更新数据的状态, 并没有真正删除
2 db.query('UPDATE USERS SET status=1 WHERE id=?', 6, (err, results) => {
3   if (err) return console.log(err.message) // 失败
4   if (results.affectedRows === 1) { console.log('删除数据成功! ') } // 成功
5 })
```



# 目录 Contents

- ◆ 数据库的基本概念
- ◆ 安装并配置 MySQL
- ◆ MySQL的基本使用
- ◆ 在项目中操作 MySQL
- ◆ 前后端的身份认证

## 5. 前后端的身份认证

### 5.1 Web 开发模式

目前主流的 Web 开发模式有两种，分别是：

- ① 基于服务端渲染的传统 Web 开发模式
- ② 基于前后端分离的新型 Web 开发模式





# 5. 前后端的身份认证

## 5.1 Web 开发模式

### 1. 服务端渲染的 Web 开发模式

服务端渲染的概念：服务器发送给客户端的 HTML 页面，是在服务器通过字符串的拼接，动态生成的。因此，客户端不需要使用 Ajax 这样的技术额外请求页面的数据。代码示例如下：

```
1 app.get('/index.html', (req, res) => {
2   // 1. 要渲染的数据
3   const user = { name: 'zs', age: 20 }
4   // 2. 服务器端通过字符串的拼接，动态生成 HTML 内容
5   const html = `

# 姓名: ${user.name}, 年龄: ${user.age}</h1>` 6 // 3. 把生成好的页面内容响应给客户端。因此，客户端拿到的是带有真实数据的 HTML 页面 7 res.send(html) 8 })


```

## 5. 前后端的身份认证

### 5.1 Web 开发模式

#### 2. 服务端渲染的优缺点

优点：

- ① **前端耗时少。**因为服务器端负责动态生成 HTML 内容，浏览器只需要直接渲染页面即可。尤其是移动端，更省电。
- ② **有利于SEO。**因为服务器端响应的是完整的 HTML 页面内容，所以爬虫更容易爬取获得信息，更有利于 SEO。

缺点：

- ① **占用服务器端资源。**即服务器端完成 HTML 页面内容的拼接，如果请求较多，会对服务器造成一定的访问压力。
- ② **不利于前后端分离，开发效率低。**使用服务器端渲染，则**无法进行分工合作**，尤其对于**前端复杂度高**的项目，不利于项目高效开发。

## 5. 前后端的身份认证

### 5.1 Web 开发模式

#### 3. 前后端分离的 Web 开发模式

前后端分离的概念：前后端分离的开发模式，**依赖于 Ajax 技术的广泛应用**。简而言之，前后端分离的 Web 开发模式，就是**后端只负责提供 API 接口，前端使用 Ajax 调用接口**的开发模式。



## ■ 5. 前后端的身份认证

### 5.1 Web 开发模式

#### 4. 前后端分离的优缺点

优点：

- ① **开发体验好。** 前端专注于 UI 页面的开发，后端专注于api 的开发，且前端有更多的选择性。
- ② **用户体验好。** Ajax 技术的广泛应用，极大的提高了用户的体验，可以轻松实现页面的局部刷新。
- ③ **减轻了服务器端的渲染压力。** 因为页面最终是在每个用户的浏览器中生成的。

缺点：

- ① **不利于 SEO。** 因为完整的 HTML 页面需要在客户端动态拼接完成，所以爬虫对无法爬取页面的有效信息。（解决方案：利用 Vue、React 等前端框架的 **SSR**（server side render）技术能够很好的解决 SEO 问题！）



## 5. 前后端的身份认证

### 5.1 Web 开发模式

#### 5. 如何选择 Web 开发模式

**不谈业务场景而盲目选择使用何种开发模式都是耍流氓。**

- 比如企业级网站，主要功能是展示而没有复杂的交互，并且需要良好的 SEO，则这时我们就需要使用服务器端渲染；
- 而类似后台管理项目，交互性比较强，不需要考虑 SEO，那么就可以使用前后端分离的开发模式。

另外，具体使用何种开发模式并不是绝对的，为了**同时兼顾**了**首页的渲染速度**和**前后端分离的开发效率**，一些网站采用了首屏服务器端渲染 + 其他页面前后端分离的开发模式。

## 5. 前后端的身份认证

### 5.2 身份认证

#### 1. 什么是身份认证

**身份认证** (Authentication) 又称“身份验证”、“鉴权”，是指**通过一定的手段，完成对用户身份的确认。**

- 日常生活中的身份认证随处可见，例如：高铁的验票乘车，手机的密码或指纹解锁，支付宝或微信的支付密码等。
- 在 Web 开发中，也涉及到用户身份的认证，例如：各大网站的**手机验证码登录**、**邮箱密码登录**、**二维码登录**等。



## 5. 前后端的身份认证

### 5.2 身份认证

#### 2. 为什么需要身份认证

身份认证的目的，是为了**确认当前所声称某种身份的用户，确实是所声称的用户**。例如，你去找快递员取快递，你要怎么证明这份快递是你的。

在互联网项目开发中，如何对用户的身份进行认证，是一个值得深入探讨的问题。例如，如何才能保证网站不会错误的将“马云的存款数额”显示到“马化腾的账户”上。

## 5. 前后端的身份认证

### 5.2 身份认证

#### 3. 不同开发模式下的身份认证

对于服务端渲染和前后端分离这两种开发模式来说，分别有着不同的身份认证方案：

- ① 服务端渲染推荐使用 Session 认证机制
- ② 前后端分离推荐使用 JWT 认证机制



# 5. 前后端的身份认证

## 5.3 Session 认证机制

### 1. HTTP 协议的无状态性

了解 HTTP 协议的无状态性是进一步学习 Session 认证机制的必要前提。

HTTP 协议的无状态性，指的是客户端的**每次 HTTP 请求都是独立的**，连续多个请求之间没有直接的关系，**服务器不会主动保留每次 HTTP 请求的状态**。

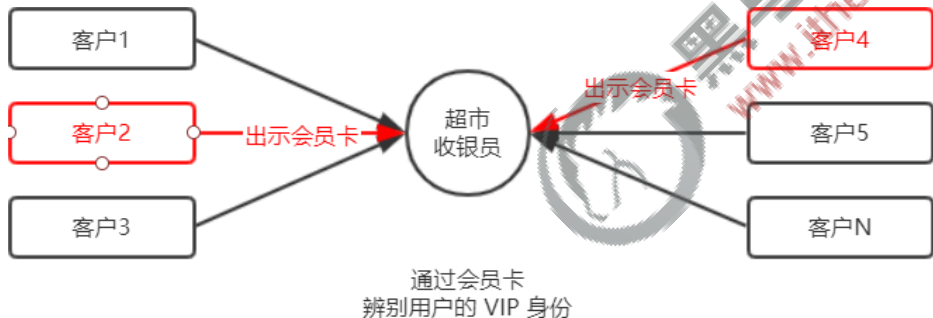


## 5. 前后端的身份认证

### 5.3 Session 认证机制

#### 2. 如何突破 HTTP 无状态的限制

对于超市来说，为了方便收银员在进行结算时给 VIP 用户打折，超市可以为每个 VIP 用户发放会员卡。



注意：现实生活中的**会员卡身份认证方式**，在 Web 开发中的**专业术语**叫做 **Cookie**。



## 5. 前后端的身份认证

### 5.3 Session 认证机制

#### 3. 什么是 Cookie

Cookie 是**存储在用户浏览器中的一段不超过 4 KB 的字符串**。它由一个**名称 (Name)**、一个**值 (Value)** 和其它几个用于控制 Cookie **有效期**、**安全性**、**使用范围**的**可选属性**组成。

不同域名下的 Cookie 各自独立，每当客户端发起请求时，会**自动把当前域名下所有未过期的 Cookie**一同发送到服务器。

**Cookie的几大特性：**

- ① 自动发送
- ② 域名独立
- ③ 过期时限
- ④ 4KB 限制

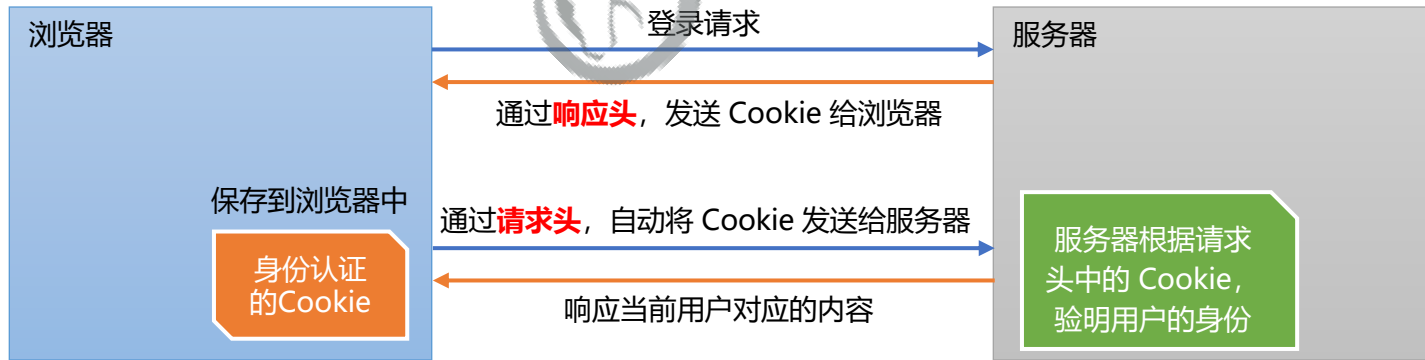
## 5. 前后端的身份认证

### 5.3 Session 认证机制

#### 4. Cookie 在身份认证中的作用

客户端第一次请求服务器的时候，服务器**通过响应头的形式**，向客户端发送一个身份认证的 Cookie，客户端会自动将 Cookie 保存在浏览器中。

随后，当客户端浏览器每次请求服务器的时候，浏览器会**自动**将身份认证相关的 Cookie，**通过请求头的形式**发送给服务器，服务器即可验明客户端的身份。

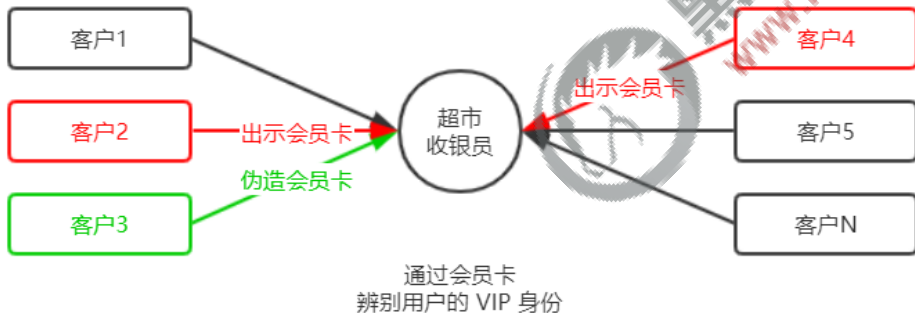


## 5. 前后端的身份认证

### 5.3 Session 认证机制

#### 5. Cookie 不具有安全性

由于 Cookie 是存储在浏览器中的，而且浏览器也提供了读写 Cookie 的 API，因此 Cookie 很容易被伪造，不具有安全性。因此不建议服务器将重要的隐私数据，通过 Cookie 的形式发送给浏览器。



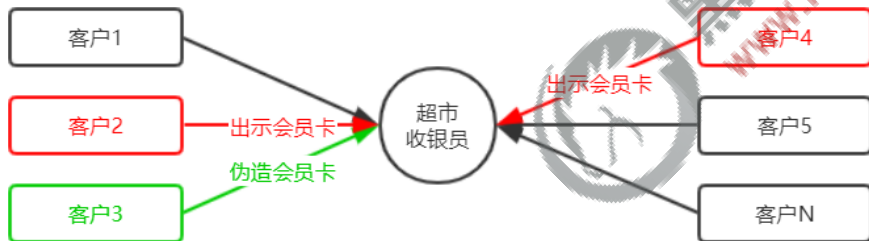
**注意：**千万不要使用 Cookie 存储重要且隐私的数据！比如用户的身份信息、密码等。

## 5. 前后端的身份认证

### 5.3 Session 认证机制

#### 6. 提高身份认证的安全性

为了防止客户伪造会员卡，收银员在拿到客户出示的会员卡之后，可以在**收银机上进行刷卡认证**。只有收银机确认存在的会员卡，才能被正常使用。



会员卡 + 刷卡认证

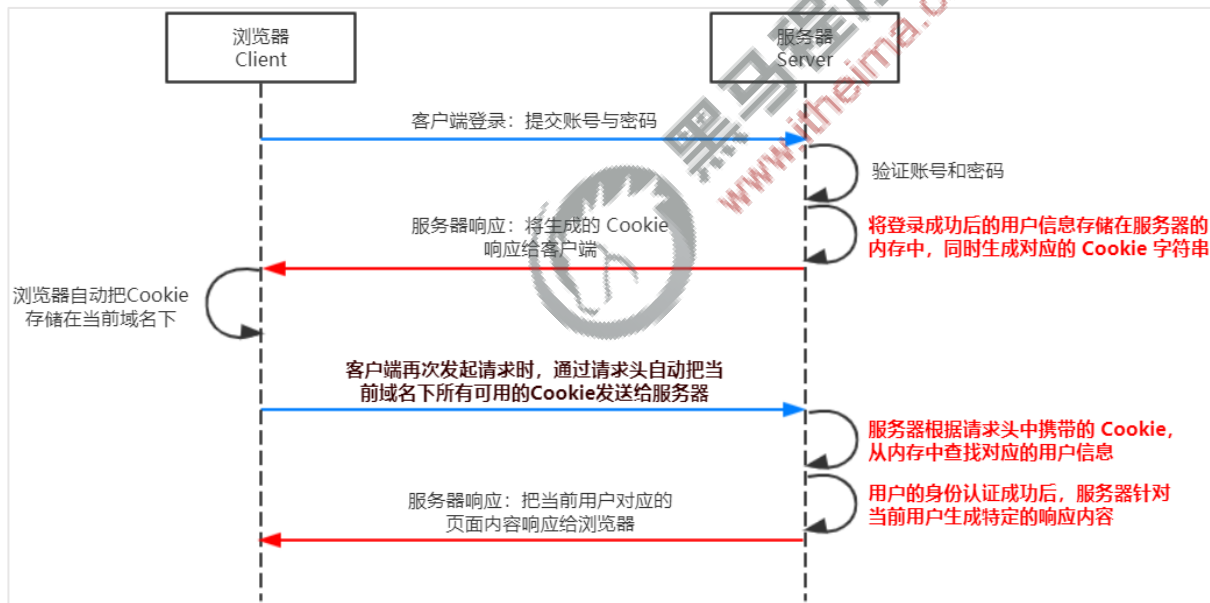
这种“会员卡 + 刷卡认证”的设计理念，就是 Session 认证机制的精髓。



## 5. 前后端的身份认证

### 5.3 Session 认证机制

#### 7. Session 的工作原理



## 5. 前后端的身份认证

### 5.4 在 Express 中使用 Session 认证

#### 1. 安装 express-session 中间件

在 Express 项目中，只需要安装 `express-session` 中间件，即可在项目中使用 Session 认证：

```
1 npm install express-session
```



## 5. 前后端的身份认证

### 5.4 在 Express 中使用 Session 认证

#### 2. 配置 express-session 中间件

express-session 中间件安装成功后，需要通过 `app.use()` 来注册 session 中间件，示例代码如下：

```
1 // 1. 导入 session 中间件
2 var session = require('express-session')
3
4 // 2. 配置 Session 中间件
5 app.use(session({
6   secret: 'keyboard cat', // secret 属性的值可以为任意字符串
7   resave: false,          // 固定写法
8   saveUninitialized: true // 固定写法
9 })))
```



## 5. 前后端的身份认证

### 5.4 在 Express 中使用 Session 认证

#### 3. 向 session 中存数据

当 express-session 中间件配置成功后，即可通过 **req.session** 来访问和使用 session 对象，从而存储用户的关键信息：

```
1 app.post('/api/login', (req, res) => {
2   // 判断用户提交的登录信息是否正确
3   if (req.body.username !== 'admin' || req.body.password !== '000000') {
4     return res.send({ status: 1, msg: '登录失败' })
5   }
6
7   req.session.user = req.body // 将用户的信息, 存储到 Session 中
8   req.session.islogin = true // 将用户的登录状态, 存储到 Session 中
9
10  res.send({ status: 0, msg: '登录成功' })
11 })
```



## 5. 前后端的身份认证

### 5.4 在 Express 中使用 Session 认证

#### 4. 从 session 中取数据

可以直接从 `req.session` 对象上获取之前存储的数据，示例代码如下：

```
1 // 获取用户姓名的接口
2 app.get('/api/username', (req, res) => {
3   // 判断用户是否登录
4   if (!req.session.islogin) {
5     return res.send({ status: 1, msg: 'fail' })
6   }
7   res.send({ status: 0, msg: 'success', username: req.session.user.username })
8 })
```



## 5. 前后端的身份认证

### 5.4 在 Express 中使用 Session 认证

#### 5. 清空 session

调用 **req.session.destroy()** 函数，即可清空服务器保存的 session 信息。

```
1 // 退出登录的接口
2 app.post('/api/logout', (req, res) => {
3   // 清空当前客户端对应的 session 信息
4   req.session.destroy()
5   res.send({
6     status: 0,
7     msg: '退出登录成功'
8   })
9 })
```

## 5. 前后端的身份认证

### 5.5 JWT 认证机制

#### 1. 了解 Session 认证的局限性

Session 认证机制需要配合 Cookie 才能实现。由于 Cookie 默认不支持跨域访问，所以，当涉及到前端跨域请求后端接口的时候，需要做很多额外的配置，才能实现跨域 Session 认证。

注意：

- 当前端请求后端接口不存在跨域问题的时候，推荐使用 Session 身份认证机制。
- 当前端需要跨域请求后端接口的时候，不推荐使用 Session 身份认证机制，推荐使用 JWT 认证机制。

## 5. 前后端的身份认证

### 5.5 JWT 认证机制

#### 2. 什么是 JWT

JWT (英文全称: JSON Web Token) 是目前最流行的跨域认证解决方案。

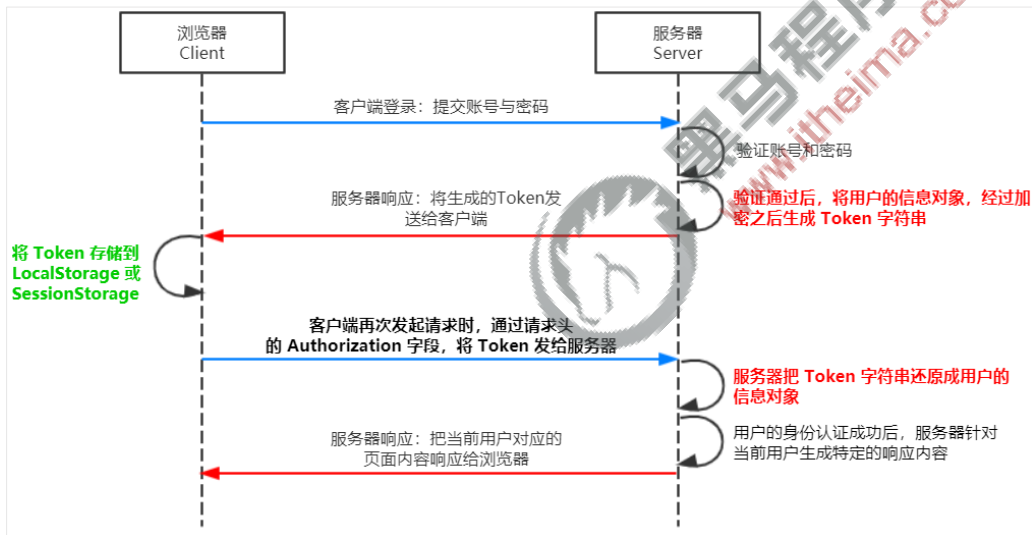




## 5. 前后端的身份认证

### 5.5 JWT 认证机制

#### 3. JWT 的工作原理



总结：用户的信息通过 Token 字符串的形式，保存在客户端浏览器中。服务器通过还原 Token 字符串的形式来认证用户的身份。



## 5. 前后端的身份认证

### 5.5 JWT 认证机制

#### 4. JWT 的组成部分

JWT 通常由三部分组成，分别是 **Header**（头部）、**Payload**（有效荷载）、**Signature**（签名）。

三者之间使用英文的 “.” 分隔，格式如下：

```
1 Header.Payload.Signature
```

下面是 JWT 字符串的示例：

```
1 eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwidXNlcm5hbWUiOiJhZG1pbiIsInBhc3N3b3JkIjoiiWibmlja25hbnWUiOiLms6X1t7Tlt7QiLCJlbWpCI6Im5pYmFiYUBpdGNhc3QuY24iLCJ1c2VyX3BpYyI6IiIsIm1hdCI6MTU3ODAzNjY4MiwizXhwIjo5NTc4MDcyNjgyfQ.Mwq7GqCxCJPK-EA8LNrtMG041lKdZ33S9KBL3XeuBxuI
```

## 5. 前后端的身份认证

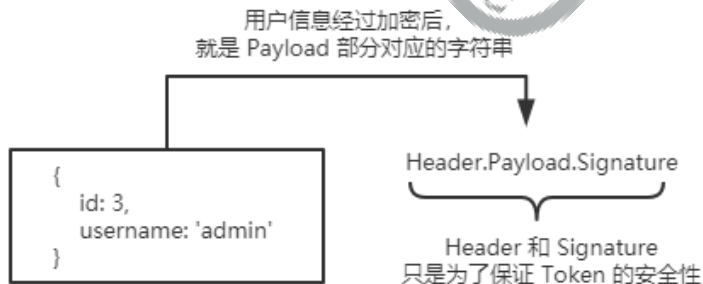
### 5.5 JWT 认证机制

#### 6. JWT 的三个部分各自代表的含义

JWT 的三个组成部分，从前到后分别是 Header、Payload、Signature。

其中：

- **Payload** 部分才是真正的用户信息，它是用户信息经过加密之后生成的字符串。
- Header 和 Signature 是安全性相关的部分，只是为了保证 Token 的安全性。



## 5. 前后端的身份认证

### 5.5 JWT 认证机制

#### 7. JWT 的使用方式

客户端收到服务器返回的 JWT 之后，通常会将它储存在 `localStorage` 或 `sessionStorage` 中。

此后，客户端每次与服务器通信，都要带上这个 JWT 的字符串，从而进行身份认证。推荐的做法是**把 JWT 放在 HTTP 请求头的 `Authorization` 字段中**，格式如下：

```
1 Authorization: Bearer <token>
```



## 5. 前后端的身份认证

### 5.6 在 Express 中使用 JWT

#### 1. 安装 JWT 相关的包

运行如下命令，安装如下两个 JWT 相关的包：

```
1 npm install jsonwebtoken express-jwt
```

其中：

- **jsonwebtoken** 用于生成 JWT 字符串
- **express-jwt** 用于将 JWT 字符串解析还原成 JSON 对象



## 5. 前后端的身份认证

### 5.6 在 Express 中使用 JWT

#### 2. 导入 JWT 相关的包

使用 **require()** 函数，分别导入 JWT 相关的两个包：

```
1 // 1. 导入用于生成 JWT 字符串的包
2 const jwt = require('jsonwebtoken')
3 // 2. 导入用于将客户端发送过来的 JWT 字符串，解析还原成 JSON 对象的包
4 const expressJWT = require('express-jwt')
```



## 5. 前后端的身份认证

### 5.6 在 Express 中使用 JWT

#### 3. 定义 secret 密钥

为了保证 JWT 字符串的安全性，防止 JWT 字符串在网络传输过程中被别人破解，我们需要专门定义一个用于加密和解密的 secret 密钥：

- ① 当生成 JWT 字符串的时候，需要使用 secret 密钥对用户的信息进行加密，最终得到加密好的 JWT 字符串
- ② 当把 JWT 字符串解析还原成 JSON 对象的时候，需要使用 secret 密钥进行解密

```
1 // 3. secret 密钥的本质：就是一个字符串
2 const secretKey = 'itheima No1 ^_^'
```



## 5. 前后端的身份认证

### 5.6 在 Express 中使用 JWT

#### 4. 在登录成功后生成 JWT 字符串

调用 `jsonwebtoken` 包提供的 `sign()` 方法，将用户的信息加密成 JWT 字符串，响应给客户端：

```
1 // 登录接口
2 app.post('/api/login', function(req, res) {
3   // ... 省略登录失败情况下的代码
4   // 用户登录成功之后，生成 JWT 字符串，通过 token 属性响应给客户端
5   res.send({
6     status: 200,
7     message: '登录成功! ',
8     // 调用 jwt.sign() 生成 JWT 字符串，三个参数分别是：用户信息对象、加密密钥、配置对象
9     token: jwt.sign({ username: userinfo.username }, secretKey, { expiresIn: '30s' })
10  })
11 })
```





## 5. 前后端的身份认证

### 5.6 在 Express 中使用 JWT

#### 5. 将 JWT 字符串还原为 JSON 对象

客户端每次在访问那些有权限接口的时候，都需要主动通过请求头中的 **Authorization 字段**，将 Token 字符串发送到服务器进行身份认证。

此时，服务器可以通过 **express-jwt** 这个中间件，自动将客户端发送过来的 Token 解析还原成 JSON 对象：

```
1 // 使用 app.use() 来注册中间件
2 // expressJWT({ secret: secretKey }) 就是用来解析 Token 的中间件
3 // .unless({ path: [/^\/api\/]/ }) 用来指定哪些接口不需要访问权限
4 app.use(expressJWT({ secret: secretKey }).unless({ path: [/^\/api\/]/ })))
```



## 5. 前后端的身份认证

### 5.6 在 Express 中使用 JWT

#### 6. 使用 req.user 获取用户信息

当 express-jwt 这个中间件配置成功之后，即可在那些有权限的接口中，使用 **req.user** 对象，来访问从 JWT 字符串中解析出来的用户信息了，示例代码如下：

```
1 // 这是一个有权限的 API 接口
2 app.get('/admin/getinfo', function(req, res) {
3   console.log(req.user)
4   res.send({
5     status: 200,
6     message: '获取用户信息成功! ',
7     data: req.user
8   })
9 })
```



## 5. 前后端的身份认证

### 5.6 在 Express 中使用 JWT

#### 7. 捕获解析 JWT 失败后产生的错误

当使用 express-jwt 解析 Token 字符串时，如果客户端发送过来的 Token 字符串**过期**或**不合法**，会产生一个**解析失败**的错误，影响项目的正常运行。我们可以通过 **Express 的错误中间件**，捕获这个错误并进行相关的处理，示例代码如下：

```
1 app.use((err, req, res, next) => {
2   // token 解析失败导致的错误
3   if(err.name === 'UnauthorizedError') {
4     return res.send({ status: 401, message: '无效的token' })
5   }
6   // 其它原因导致的错误
7   res.send({ status: 500, message: '未知错误' })
8 })
```



黑马程序员

[www.itheima.com](http://www.itheima.com)

传智播客旗下高端IT教育品牌