数据库系统实现实验二（第8周）

实验目的：

1. 安装miniOB编译环境，能够建立编译生成miniOB的执行代码，能够启动服务端、客户端，并在客户端运行自己的SQL命令测试用例。
2. 安装IDE（VScode）根据手册、视频、代码解析指导，完成drop table 试题17；

熟悉github,gitee,git,linux环境操作命令

1. 根据视频、代码解析指导，完成增加新类型date的题目14。

实验平台：

<https://open.oceanbase.com/train?questionId=200001> 训练营网址

<https://open.oceanbase.com/activities/4921877> 从0到1数据库内核实战教程视频

实验内容、步骤：

一、安装miniOB编译环境，能够建立编译生成miniOB的执行代码，能够启动服务端、客户端，并在客户端运行自己的SQL命令测试用例。

**编译环境可用多种方式搭建，vm , wsl, docker**, 以下以Docker为例介绍。

1. **搭建** Docker **环境**

Docker 是一个开源的应用容器引擎，基于 [Go语言](https://www.runoob.com/go/go-tutorial.html) 并遵从 Apache2.0 协议开源。Docker 可以让开发者打包他们的应用以及依赖包到一个轻量级、可移植的容器中，然后发布到任何机器上，也可 以实现虚拟化。

**（**1**）安装**docker

请参考如下教程安装Docker，[如何在Windows上使用Docker开发miniob](https://github.com/oceanbase/miniob/blob/main/docs/how_to_dev_miniob_by_docker_on_windows.md)

**（**2**）挂载本地目录到** docker **镜像中**

为了更加方便安全和优雅的进行后续开发，通常我们**将代码保存至自己本地电脑上，Docker 中运**

**行的容器仅作为代码编译的环境**。开发时，**代码存放于本地，如果容器出现什么异常，自己最重要 的数据不会丢失**。这里使用 **mount** 挂载的方式，**将一个本地目录映射到 Docker 容器中**。

安装完毕之后，可以使用 **Windows cmd 或者 powershell 环境** 执行如下命令，下载并运行

miniob 镜像。

SHELL

docker run -d --name miniob --privileged -v $PWD/miniob:/root/miniob

oceanbase/miniob

其中**-d** 参数是后台运行容器，并返回容器 ID， **--name miniob** 参数是重命名容器名称为

miniob，这里也可以用其他名称命名， **--privileged** 参数是为了方便在容器中进行调试， **-v**

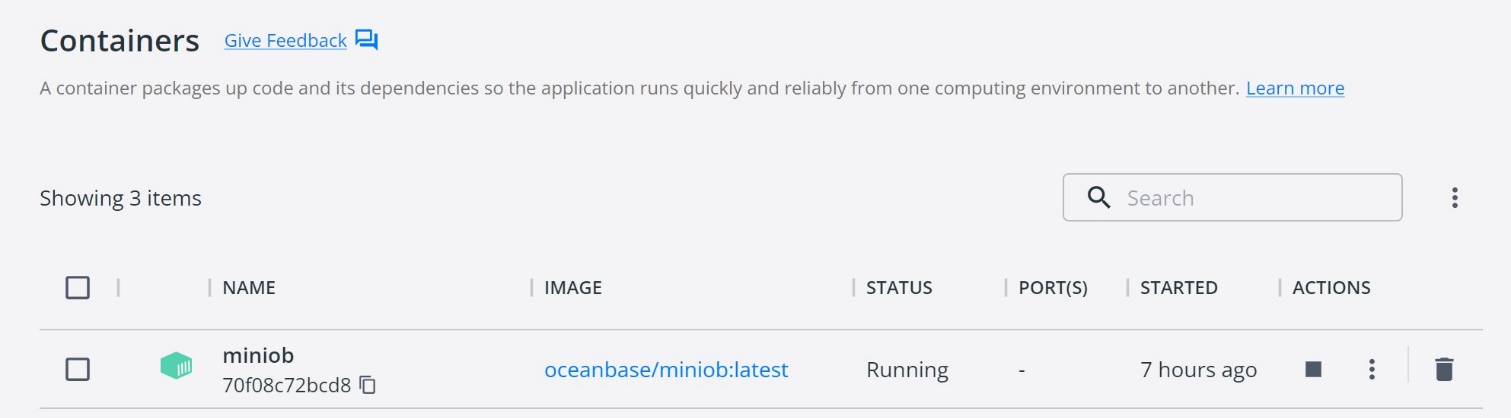
**$PWD/miniob:/root/miniob** 参数是将本地的 **$PWD/miniob** 目录映射到容器中的

**/root/miniob** 目录，这里假设 **$PWD/miniob** 是你本地代码存放的位置，需要按照实际目录来调

整。本手册默认代码存放位置为 **d:\Download\miniob** , **$PWD** 应为 **d:\Download** ，

**oceanbase/miniob** 参数就是镜像的名称，Docker 会自动从云端拉取该镜像的最新版本。

此时可以看到 Docker Desktop 中已经有一个名为 miniob 的容器在运行。



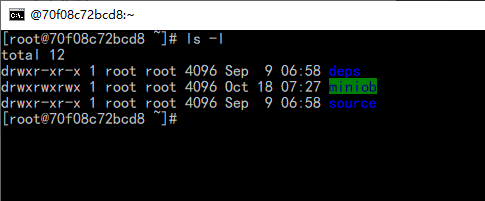
**（**3**）使用镜像内的** bash **命令行**

运行下面的命令可以进入到容器并进行开发，这个命令可以在多个终端窗口上执行，开启多个窗口方便操作。

docker exec -it miniob bash

SHELL

此时，我们应该可以看到本地目录下，是这样目录结构。

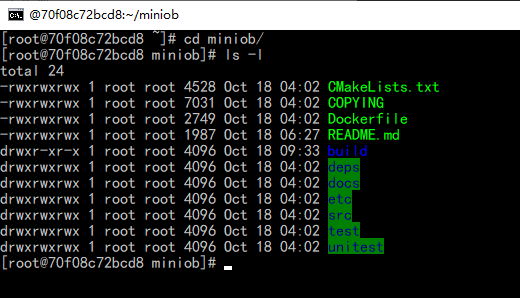


其中**/deps** 和 **/source** 是该镜像中自带的源码目录， 而反色显示的**/miniob** 是用挂载方式映射 到容器中的本地目录的软链接，在容器可以直接使用 **cd** 命令进入到 miniob 目录中查看文件。

**（**4**）创建**./build **目录存放编译产生的文件**

进入到该目录中，为了存放编译之后生成的文件，需要使用 **mkdir ./build** 命令创建 build 目

录。



此时就可以通过**cmake** 、 **make** 等终端命令编译代码了。

#编译

mkdir build （如上面已做，此bu可跳过）

cd build

cmake .. -DDEBUG=ON

# 进入容器，开发调试

docker exec -it miniob bash #以后会反复使用此命令

#运行服务端：

./bin/observer -s miniob.sock -f ../etc/observer.ini & #使用linux socket方式启动，生成observer.log日志文件。

#运行客户端：

./bin/obclient -s miniob.sock

#测试SQL命令

show tables；

desc `table name`;

create table `table name` (`column name` `column type`, ...);

create index `index name` on `table` (`column`);

insert into `table` values(`value1`,`value2`);

update `table` set column=value [where `column`=`value`];

delete from `table` [where `column`=`value`];

select [ \* | `columns` ] from `table`;

exit #退出

<https://blog.csdn.net/chrisy521/article/details/127142293> 参考2.MiniOB概述和环境搭建

<https://open.oceanbase.com/activities/4921877>

2 **使用** Visual Studio Code **调试代码**

上面的方法，每次修改代码之后，都需要输入编译的一系列指令，比较繁琐。所以接下来我们利用 vscode，使用图形 化界面和丰富的扩展插件调试代码。

使用集成开发环境（IDE）进行数据库系统代码调试，这里以最为常用的 Visual Studio Code

（简称 vscode）为例进行演示。下载地址：  [https://code.visualstudio.com/Download](%20https://code.visualstudio.com/Download%20) ，参考

教程：  [https://code.visualstudio.com/learn](%20https://code.visualstudio.com/learn%20) 。

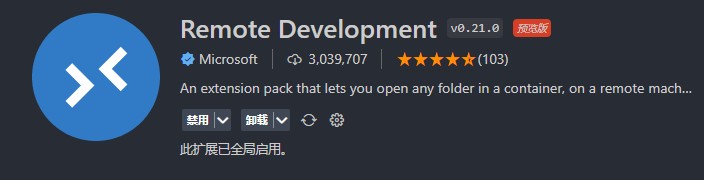
**（**1**）本地安装扩展**

启动 vscode 后，在“扩展”中安装 Docker，C/C++，# C/C++ Extension Pack，Remote

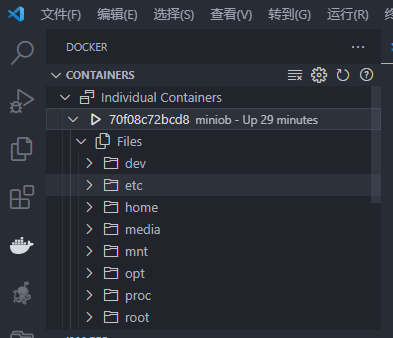
Development 等插件，安装方法是在“扩展”-“搜索框”中搜索相应名称，选择“安装”即可。



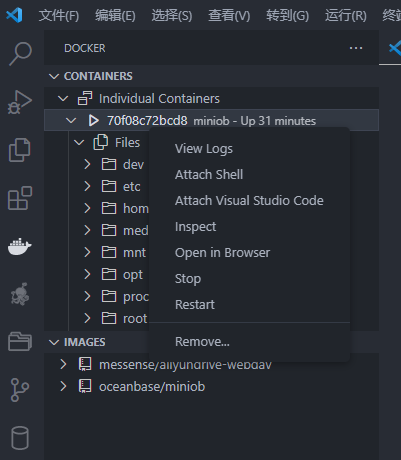




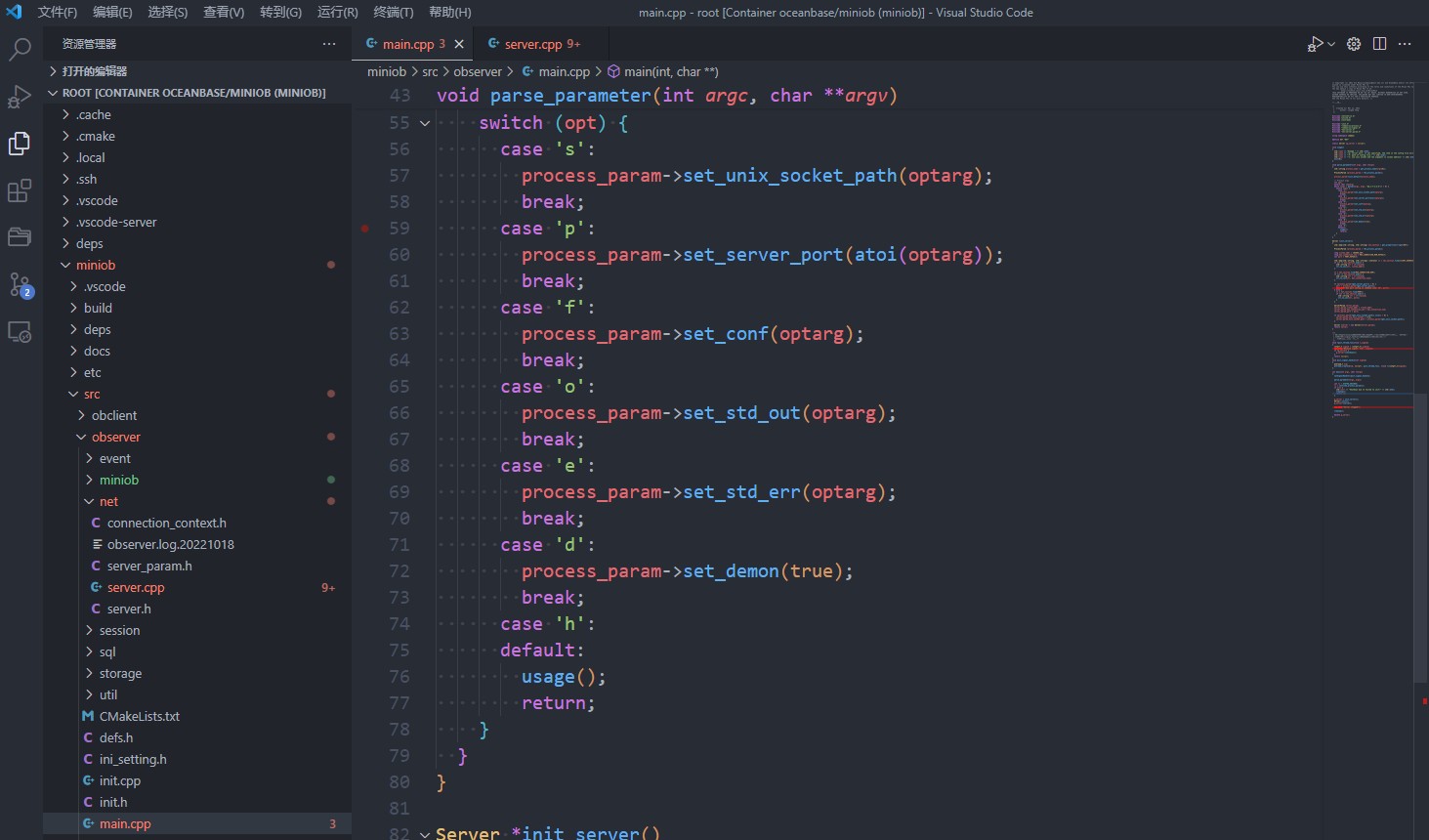
如果成功安装了 docker 环境，此时就可以看到运行的 miniob 容器以及文件列表。

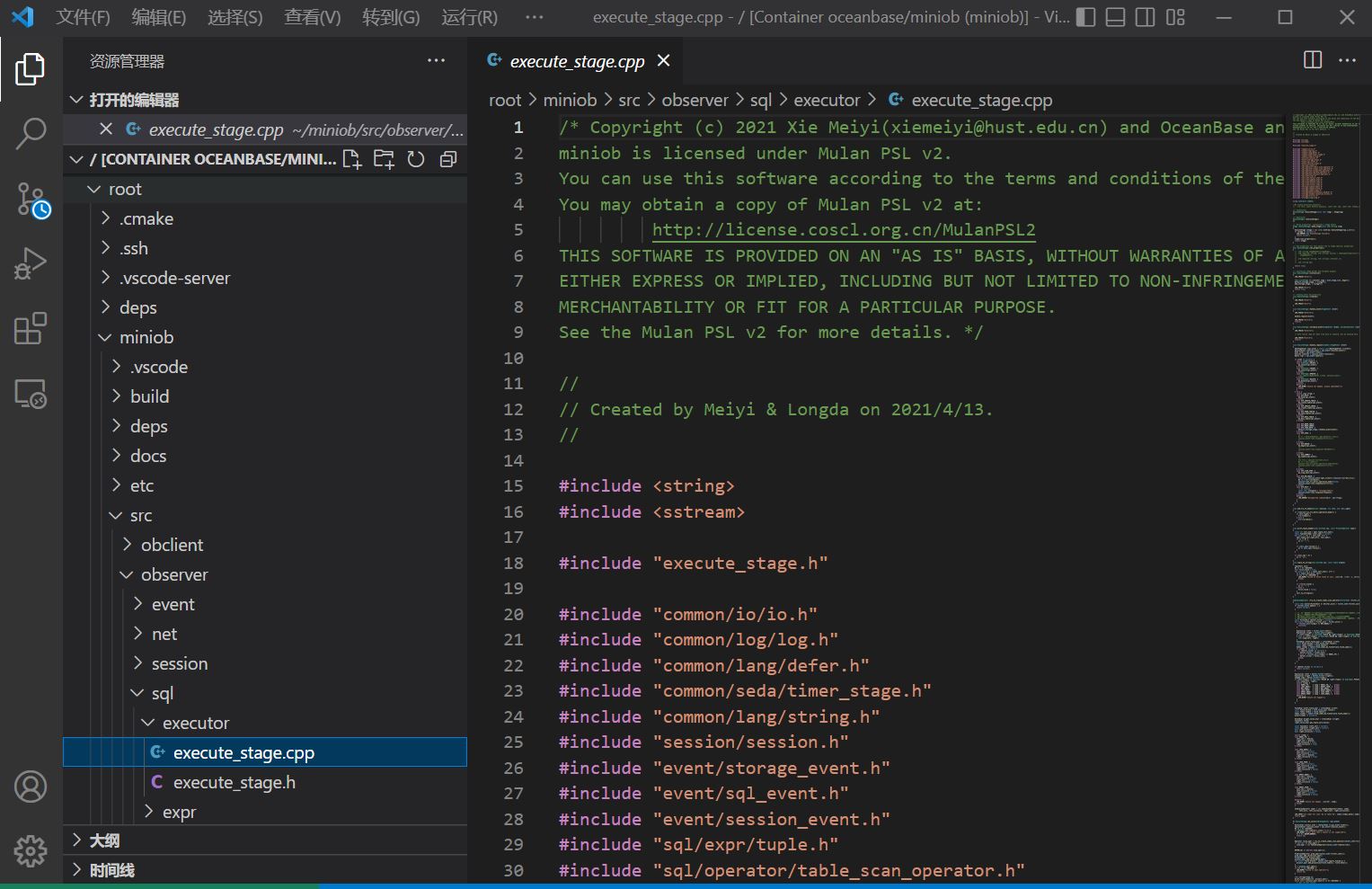


在容器上右键选择 Attach Visual Studio Code，会启动一个新窗口。



此时，我们就利用 vscode 进入到 docker 镜像中，可以开发并调试代码了。从文件列表中，我们可 以直接打开对应的源码文件进行编辑。





**（**2**）容器内安装扩展**

为了在 vscode 中编译代码，我们需要在容器内的“扩展”中安装 C/C++，# C/C++ Extension Pack

等插件，用于编译 C++程序。





注意，安装成功后，插件下面会提示“已在 Container 中启动扩展”。以下我们所有操作都是在 容器内进行。

**（**3**）配置编译和运行命令（**vscode **任务）**

在容器内/root 目录下创建 .vscode 目录（注意，以 . 为前缀的目录是隐藏目录）, 用于存放对本项 目的各种vscode配置文件。

在 .vscode 目录中，创建 tasks.json 文件，将以下配置命令拷贝粘贴并保存。

{

"options": {

"cwd": "${workspaceFolder}/miniob/build"

},

"tasks": [

{

"label": "cmake", "command": "cmake", "args": [

"-DDEBUG=ON",

".."

]

},

{

"label": "make", "command": "make"

},

{

"label": "CMake Build", "dependsOn": [

"cmake",

"make"

]

},

{

"label": "Get pid of observer", "type": "shell",

"command":"ps aux | grep observer"

},

{

"label": "Run observer", "type": "shell",

"command":"/root/miniob/build/bin/observer -s miniob-test.sock -f

/root/miniob/etc/observer.ini"

},

{

"label": "Run obclient", "type": "shell",

"command":"/root/miniob/build/bin/obclient -s miniob-test.sock"

}

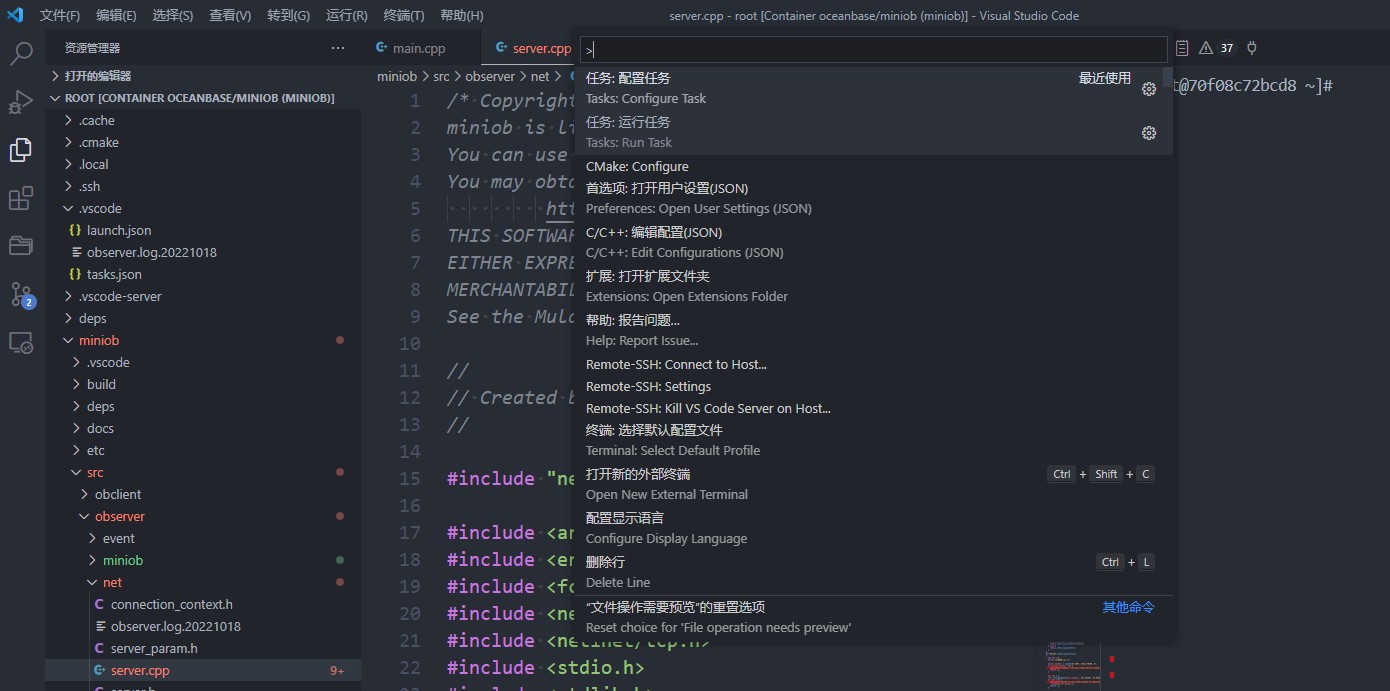
],

"version": "2.0.0"

}

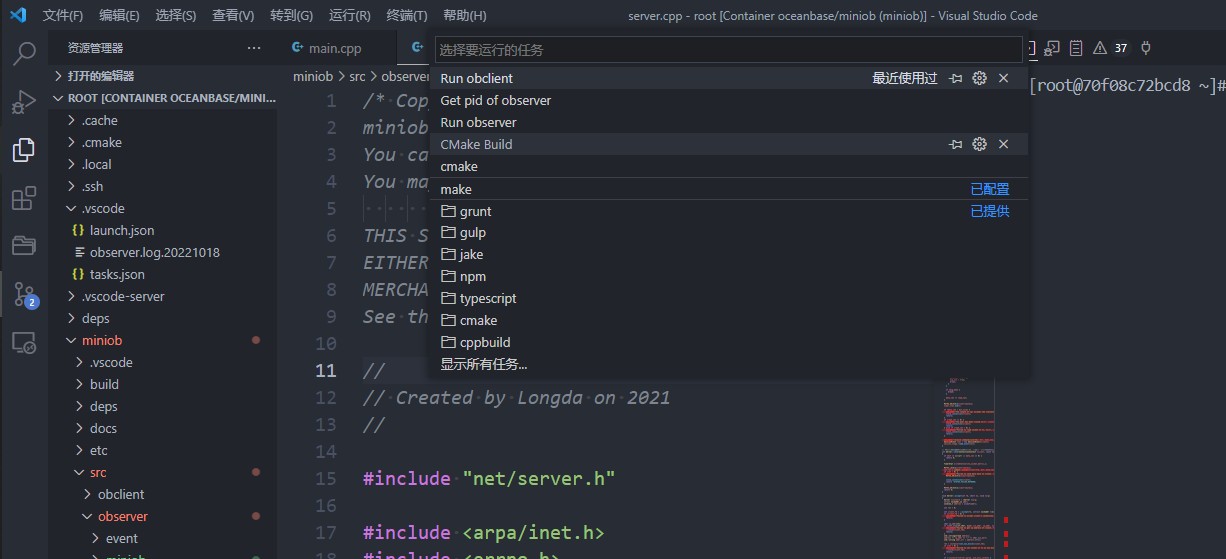
完成以上步骤后，在本项目中就配置了 5 个快捷任务，使用 **Ctrl + Shift + P** 快捷键调出快捷

命令，选择“任务：运行任务”。

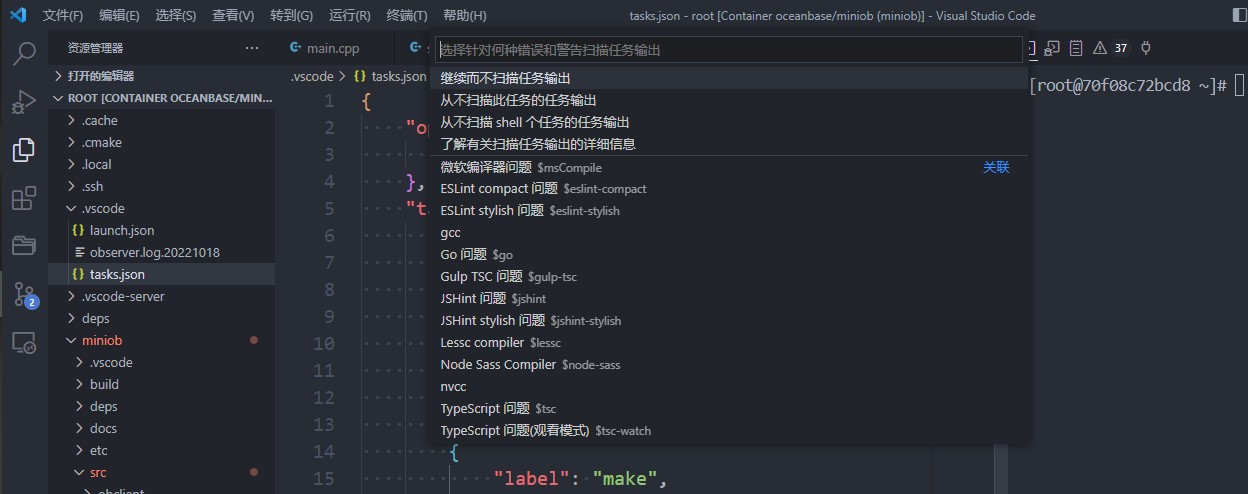


此时就可以看到刚刚配置好的任务列表，其中 **Cmake Build** 任务的前置任务分别是 **cmake** 和

**make** ，依次在 miniob 目录下编译并链接源文件。



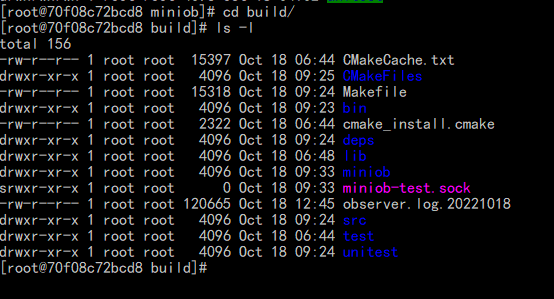
运行 **Cmake Build** 任务，弹出的选项中选择“**继续而不扫描任务输出**”。



等待一段时间，如果代码无误，即可编译成功。



执行完以上任务，可以在 build 目录下看到生成的文件和目录。其中可执行文件在 bin 目录下。

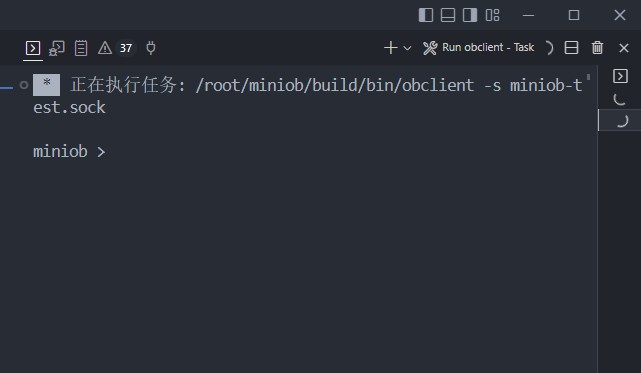


编译好的文件就可以运行调试，使用 **Run observer** 就可以运行 miniob 的 observer 命令，启动

一个 miniob 数据库服务器。启动成功后，有如下提示。



使用 **Run obclient** 可以运行 miniob 的 obclient 命令，在一个新的终端启动数据库客户端，客 户端可以直接连接刚启动的 miniob数据库。



注 1：要先启动 observer，再启动 obclient 才能成功连接。

注 2：observer的启动过程是通过建立一个 miniob-test.sock 的套接字软链接以及通过指定../../etc/observer.ini 配置文件启动的。

此时，我们已经可以运行由源代码编译完成的数据库程序了。

通过 obclient 登陆数据库，完成基本的创建表、查看表的测试语句。

SQL测试，同上。

**（**4**）配置** Debug **环境**

vscode 最强大的工具应该是其 debug 工具，可以方便使用图形界面的形式展示 debug过程中的各

种参数变量，提高开发效率。

首先，需要在 .vscode目录中创建 launch.json 文件，并将以下配置信息拷贝粘贴并保存。

{

"version": "0.2.0", "configurations": [

{

"name": "(gdb) Attach", "type": "cppdbg", "request": "attach",

"program": "${workspaceFolder}/miniob/build/bin/observer", "MIMode": "gdb",

"setupCommands": [

{

"description": "为 gdb 启用整齐打印", "text": "-enable-pretty-printing",

"ignoreFailures": true

},

{

"description": "将反汇编风格设置为 Intel",

"text": "-gdb-set disassembly-flavor intel",

"ignoreFailures": true

}

]

},

]

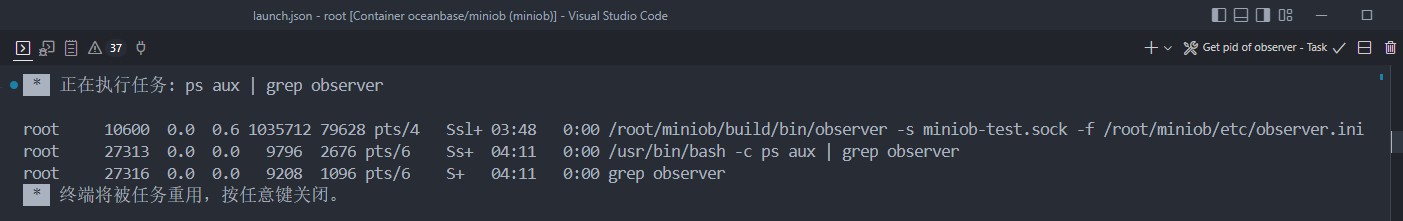
}

运行和调试的 Attach 功能可以跟踪一个运行中的进程，从而对其进行调试。**数据库系统的调试过**

**程往往是通过运行一个编译好的数据库程序，然后利用 Attach 方法跟踪其进程并在指定代码位置**

**设置断点，从断点处逐步调试编写好的程序**。

所以使用 Attach 功能之前，我们需要获得运行中的 observer 的进程号，此时继续使用上文提到的快捷任务。**Ctrl + Shift+ P** 调出快捷任务，运行 **Get pid of observer** ，获得运行中的 observer 进程号。一般情况下会得到多个结果，选择运行数据库服务的那条命令对应的行（截图中 为第一行），找到进程号为 **10600** 。该进程号在每次启动 observer 进程时都会变化，所以需要每 次 Attach 之前都查询确定。



此时，在运行和调试界面下就可以看到 **(gdb) Attach** 选项。

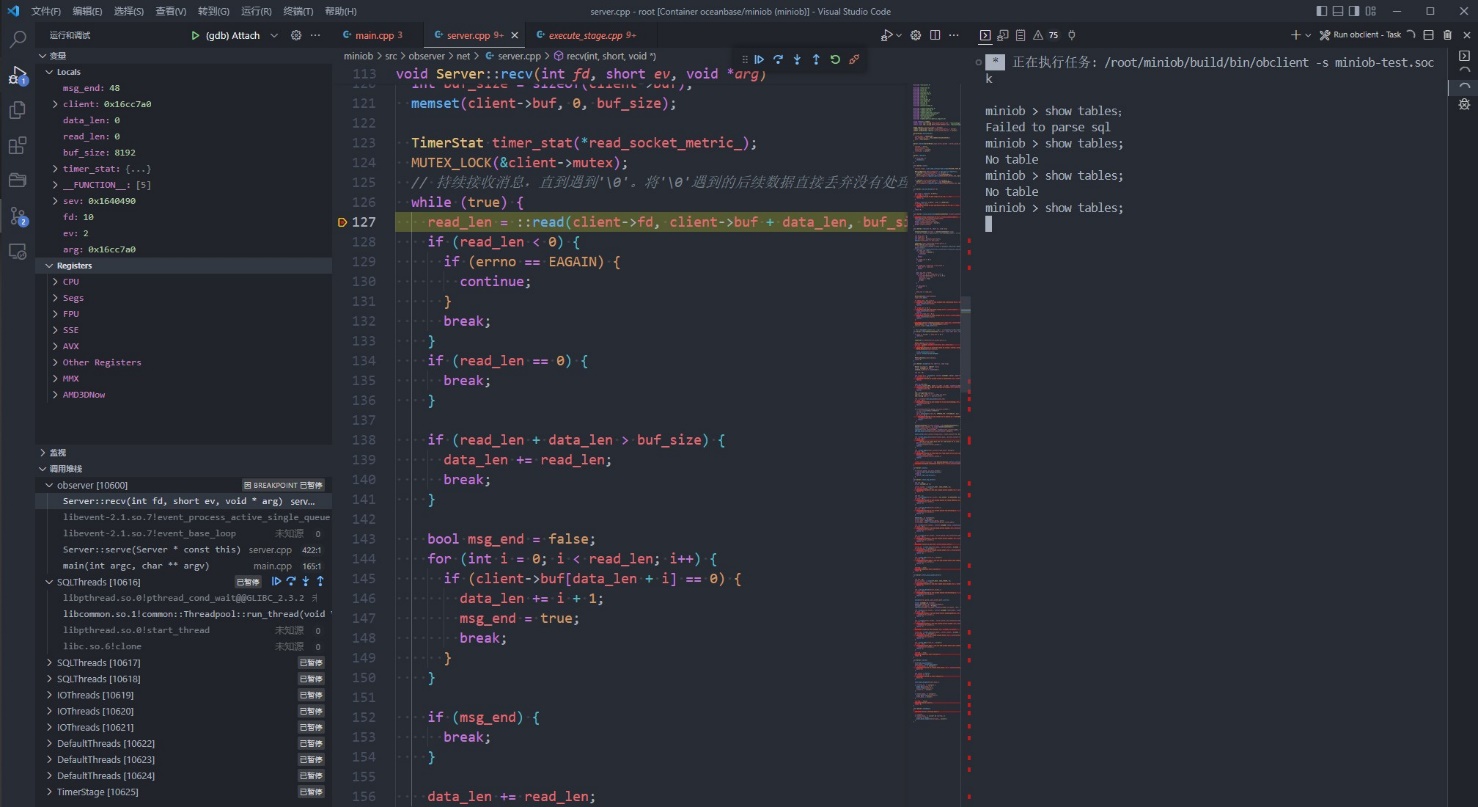


点击绿色箭头，在弹出的对话框中填入刚获知的进程号 10600，就可以进入 Attach 调试状态。



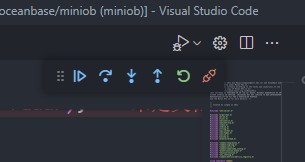
为了便于演示，在 miniob/src/observer/net/server.cpp 文件中的 Server: recv 方法中设置断点，

Attach 之后就进入跟踪状态。此时我们通过 obclient（上文启动过的客户端）执行任意 SQL 语句，就会在断点处暂停执行。



可以在屏幕左侧导航窗口看到此时变量、堆栈等信息，同时客户端也处于阻塞状态，并没有回显结果，表示成功对代码进行Attach 单步调试。

使用 **F10 ，F11 ，Shift + F11** 快捷键可以单步执行、单步跳过或者单步跳出，也可以点击相应 图标实现单步调试功能。



2.3**提测代码**

三、drop table 习题17,

参考“从0-1”视频8 <https://open.oceanbase.com/activities/4921877>

代码解析： <https://oceanbase-partner.github.io/lectures-on-dbms-implementation/miniob-drop-table-implementation.html>

四、添加date字段

参考“从0-1”视频14 <https://open.oceanbase.com/activities/4921877>

代码解析： <https://oceanbase-partner.github.io/lectures-on-dbms-implementation/miniob-date-implementation.html>

<https://oceanbase-partner.github.io/lectures-on-dbms-implementation/miniob-test-comment-date.html>