docker数据持久化（mysql持久化）

一

1、什么是数据持久化?

持久化是将程序数据在持久状态和瞬时状态间转换的机制。通俗的讲，就是瞬时数据（比如内存中的数据，是不能永久保存的）持久化为持久数据（比如持久化至数据库中，能够长久保存）。

2、理解?

持久化是一种对象服务，就是把内存中的对象保存到外存中，让以后能够取回。【当然我们这里是持久化到阿里云服务器中】

3、两个层面?

> 应用层
>
> 如果关闭(shutdown)你的应用然后重新启动则先前的数据依然存在。
>
> 系统层
>
> 如果关闭(shutdown)你的系统（电脑）然后重新启动则先前的数据依然存在

4、持久化目的?

> - 通过持久化技术可以减少访问数据库数据次数，增加应用程序执行速度;
> - 代码重用性高，能够完成大部分数据库操作;
> - 松散耦合，使持久化不依赖于底层数据库和上层业务逻辑实现，更换数据库时只需修改配置文件而不用修改代码。

二

1、拉取mysql镜像

```
[root@Bob Bob]# docker pull mysql
```

2、查看镜像

```
[root@Bob Bob]# docker images
REPOSITORY          TAG                 IMAGE ID
CREATED             SIZE
mysql               latest              990386cbd5c0
5 days ago          443MB
```

3、运行容器

```
[root@Bob Bob]# docker run -p 3306:3306 -v
$PWD/data:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=123456 -d
mysql
8f54b8da694c5276ec83bb65241a16e458963265554a44163f08ca1316
4ffc94
```

4、查看运行的容器

```
[root@Bob Bob]# docker ps
CONTAINER ID        IMAGE               COMMAND
       CREATED             STATUS              PORTS
                      NAMES
8f54b8da694c        mysql               "docker-
entrypoint.s…"   4 seconds ago       Up 3 seconds
0.0.0.0:3306->3306/tcp, 33060/tcp   amazing_cartwright
```

5、进入容器

```
[root@Bob Bob]# docker exec -it 8f54b8da694c /bin/bash
root@8f54b8da694c:
```

6、登录数据库

```
root@8f54b8da694c:/# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.16 MySQL Community Server - GPL

Copyright (c) 2000, 2019, Oracle and/or its affiliates.
All rights reserved.

Oracle is a registered trademark of Oracle Corporation
and/or its
affiliates. Other names may be trademarks of their
respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the
current input statement.
```

7、展示数据库

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.01 sec)
```

8、创建demo的数据库

```
mysql> create database demo;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| demo               |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
```

9、退出数据库+退出容器

```
mysql> exit
Bye
root@8f54b8da694c:/# exit
exit
[root@Bob Bob]# docker ps -a
CONTAINER ID        IMAGE               COMMAND
       CREATED             STATUS              PORTS
                        NAMES
8f54b8da694c        mysql               "docker-
entrypoint.s…"   13 minutes ago      Up 13 minutes
0.0.0.0:3306->3306/tcp, 33060/tcp   amazing_cartwright
```

10、停止容器并重新运行容器

```
[root@Bob Bob]# docker stop 8f54b8da694c
8f54b8da694c
[root@Bob Bob]#docker run -p 3306:3306 -v
$PWD/data:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=123456 -d
mysql
8e8a8ab2f194b179ebe32e17a2ac28404e62ef92ff795834f48e7a7a9b
7548e0
[root@Bob Bob]# docker ps
CONTAINER ID        IMAGE               COMMAND
       CREATED             STATUS              PORTS
                       NAMES
8e8a8ab2f194        mysql               "docker-
entrypoint.s…"   20 seconds ago      Up 19 seconds
0.0.0.0:3306->3306/tcp, 33060/tcp   peaceful_brattain
```

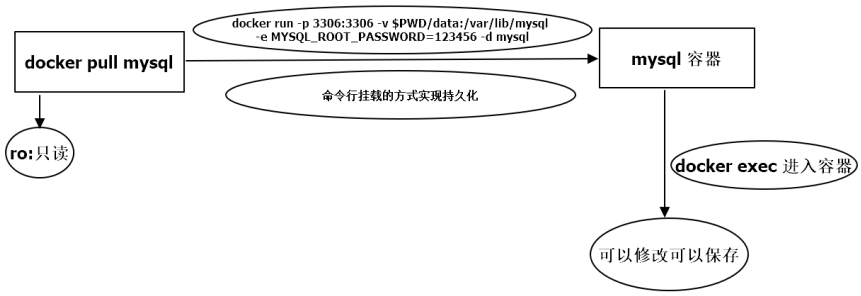
11、进入容器并重新登录数据库

```
[root@Bob Bob]# docker exec -it 8e8a8ab2f194 /bin/bash
root@8e8a8ab2f194:/# mysql -u root -p
Enter password:
```

12、展示数据库

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
  demo
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)
```

综述：



此时你再看下你本地目录下面的 data目录，就会发现多出了很多文件：

```
[root@Bob ~]# cd data
[root@Bob data]# ll
总用量 177164
-rw-r-----. 1 polkitd input       56 5月  15 10:26
auto.cnf
-rw-r-----. 1 polkitd input  3091158 5月  15 10:26
binlog.000001
-rw-r-----. 1 polkitd input      384 5月  15 10:29
binlog.000002
-rw-r-----. 1 polkitd input      155 5月  15 10:30
binlog.000003
-rw-r-----. 1 polkitd input       48 5月  15 10:30
binlog.index
-rw-------. 1 polkitd input     1676 5月  15 10:26 ca-
key.pem
-rw-r--r--. 1 polkitd input     1112 5月  15 10:26 ca.pem
-rw-r--r--. 1 polkitd input     1112 5月  15 10:26 client-
cert.pem
-rw-------. 1 polkitd input     1680 5月  15 10:26 client-
key.pem
-rw-r-----. 1 polkitd input     3472 5月  15 10:29
ib_buffer_pool
-rw-r-----. 1 polkitd input 12582912 5月  15 10:30 ibdata1
-rw-r-----. 1 polkitd input 50331648 5月  15 10:30
ib_logfile0
-rw-r-----. 1 polkitd input 50331648 5月  15 10:26
ib_logfile1
-rw-r-----. 1 polkitd input 12582912 5月  15 10:30 ibtmp1
drwxr-x---. 2 polkitd input      187 5月  15 10:30
#innodb_temp
drwxr-x---. 2 polkitd input      143 5月  15 10:26 mysql
-rw-r-----. 1 polkitd input 29360128 5月  15 10:30
mysql.ibd
drwxr-x---. 2 polkitd input     8192 5月  15 10:26
performance_schema
```

```
-rw-------. 1 polkitd input     1680 5月  15 10:26
private_key.pem
-rw-r--r--. 1 polkitd input      452 5月  15 10:26
public_key.pem
-rw-r--r--. 1 polkitd input     1112 5月  15 10:26 server-
cert.pem
-rw-------. 1 polkitd input     1676 5月  15 10:26 server-
key.pem
drwxr-x---. 2 polkitd input       28 5月  15 10:26 sys
drwxr-x---. 2 polkitd input        6 5月  15 10:27
test_docker
-rw-r-----. 1 polkitd input 12582912 5月  15 10:30
undo_001
-rw-r-----. 1 polkitd input 10485760 5月  15 10:30
undo_002
```