1. 项目说明：

(1) 环境和依赖项：

Windows10

Python3.8 Flask2.0.1、requests

C++11 pthread.h

(2) 项目目录结构：

KVDB.cpp：数据库引擎

service.py：数据库 http 服务程序，调用 KVDB 引擎

runner.py：客户端程序，启动后支持用户输入指令

test/*.py：各功能测试用例

(3) 程序构建：

使用 C++11 和 pthread.h 库将 KVDB.cpp 编译成 KVDB.exe 并放在与 service.py 相同目录即可

(4) 程序运行和使用方法：

共有三种使用方法：

•先使用 python 运行 service.py，再运行各种 test.py，test.py 中的 cmds 变量保存了各个待测试的指令，指令的输出结果会打印在 test.py 标准输出中。

• 先使用 python 运行 service.py，再运行 runner.py，在 runner.py 中输入想要执行的指令，如 PUT A 1，按下回车输入完成后，出现执行结果。

• 直接运行 KVDB.exe，并使用<clientId> <commands>的格式将指令输入到程序中，按下回车输入完成后，出现执行结果。

3. 程序架构与通信协议

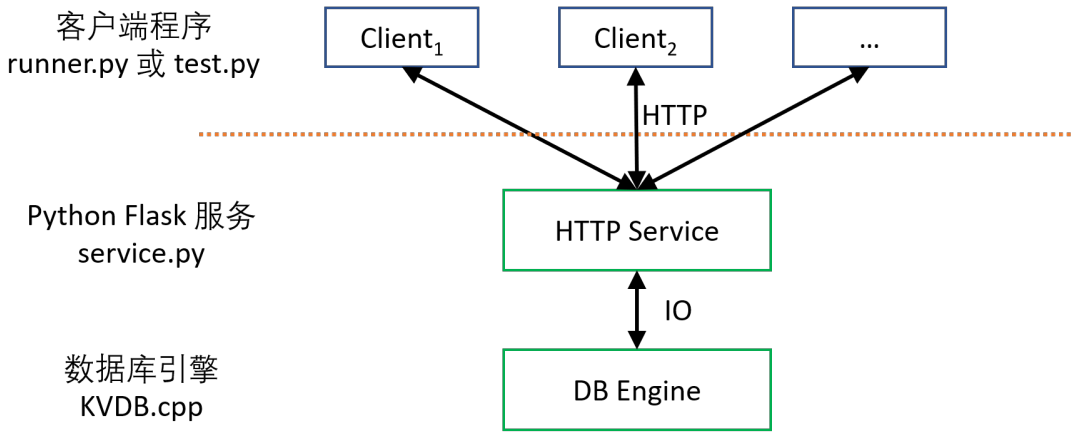

图 3.1 程序框架

在图 3.1 中，runner.py 通过标准输入接收用户的指令，支持以下指令：

- PUT <key> <value> 添加一个键值对，如 PUT A 1
- GET <key> 根据 key 获取一个值，如 GET A
- DEL <key> 删除一个键值对，如 DEL A

- BEGIN 开始一个 transaction
- COMMIT 提交当前 transaction
- ABORT 回滚当前 transaction

runner.py 在启动时，会生成一个独一无二的 clientId，并用 HTTP 协议向 HTTP Service 发送如下格式内容：

- <clientId> <commands> 如 id1 PUT A 1
- <clientId> DISCONNECT 断开一个 client 连接，如 id1 DISCONNECT

HTTP Service 会开启一个 DB Engine 进程，具体而言，service.py 中使用 Popen 打开 KVDB.exe，并使用 PIPE 绑定其 IO，将指令输入给 KVDB 并从其输出中读取结果，返回给客户端。

DB Engine 会为每一个 clientId 单独增加一个线程处理其相关请求，并在接收 DISCONNECT 指令后关闭线程。

4.代码功能细节

4.1 service.py

service.py 是一个基于 flask 的 http 服务，启动时会开启一个 KVDB.exe 的子进程，并使用 PIPE 绑定它的 IO，开启 MessageReader 线程不断监听 KVDB 的输入，并保存到消息队列列表 response_dict 中，以 clientId 作为 key，对应消息队列作为 value。在接收客户端 POST 请求后，将请求指令通过标准输入发送给 KVDB.exe，并以 timeout=60 秒的上限等待返回结果，通过 http 返回给客户端。

4.2 runner.py/test*.py

runner.py/test*.py 使用 requests 库向 service.py 服务（默认地址端口为 127.0.0.1:5000）发送 POST 请求，请求格式为：

- <clientId> <commands> 如 id1 PUT A 1
- <clientId> DISCONNECT 断开一个 client 连接，如 id1 DISCONNECT

发送请求后等待结果返回，并打印到标准输出中。

其中 runner.py 循环地接收用户输入的指令，test*.py 将指令硬编码到 cmds 字符串列表中，每条指令是一个字符串。

4.3 KVDB.cpp

- **支持磁盘：**

每 timeToDisk=10 时间落一次盘，将 data 变量拷入 data.kvdb 中。

- **MVCC 多版本：**

内存中有 data 原始数据与 mvcc_data 副本，在 transaction 发生写操作时，

锁定 mvcc_data 副本，并在 mvcc_data 副本中读写；如果是只读操作，则在原始数据中读取。

- **支持 ACID：**

Atomicity：对于含写 transaction，在 COMMIT 时将 MVCC 副本原子地写回原始数据中，在 ABORT 时将 MVCC 副本恢复到原始数据状态，代码中直接使用变量覆盖替代了 undo log，优点是易于实现，缺点是在数据量大的时候效率低。

Consistency：判断 key 唯一、判断 value 不超过 65536

Isolation：只读、读写 transaction 分离，读写在 MVCC 副本中，只读在原始数据中，提高多读少写的效率。

Durability：每隔 timeToDisk 落盘一次并清空 redo log；每个 transaction 执行成功时记录 redo log，在数据库启动时检查 redo log 的内容并进行重放。

- **多进程客户端并行、多线程服务：**

客户端(runner.py 或 test*.py)可以无限地并行启动并连接 HTTP 服务、执行指令。每个客户端会在 KVDB 引擎内单独运行一个线程进行服务。

5.测试结果

运行 service.py 后，完成下列测试。

5.1 test1.py

```
指令序列："PUT A 3","PUT B 4","GET A","GET B","DEL A","DEL B","PUT A 5","GET A","GET B","PUT B 5","GET B"
```

```
输出结果：
    PUT success
    PUT success
    3
    4
    DEL success
    DEL success
    PUT success
    5
    element not exist
    PUT success
    5
```

结果正确。

5.2 test2.py

运行 test1.py 后关闭服务，运行 test2.py

```
指令序列："GET A","GET B"
```

```
输出结果：
```

```
5
5
```

结果正确。

5.3 test3_1.py、test3_2.py、test3_3.py

同时执行三个文件，每个文件循环执行指令 10 次

test3_1.py 结果：

```
指令序列(循环10次):"PUT A 1","PUT B 1","PUT A (A+1)","PUT
B (B+1)","GET A","DEL A","GET B","DEL B"
PUT success
PUT success
PUT success
PUT success
2
DEL success
2
DEL success
```

test3_2.py 结果：

```
指令序列(循环10次):"PUT C 1","PUT D 1","PUT C (C+1)","PUT
D (D+1)","GET C","DEL C","GET D","DEL D"
PUT success
PUT success
PUT success
PUT success
2
DEL success
2
DEL success
```

test3_3.py 结果：

```
指令序列(循环10次):"PUT E 1","PUT F 1","PUT E (E+1)","PUT
F (F+1)","GET E","DEL E","GET F","DEL F"
PUT success
PUT success
PUT success
PUT success
2
DEL success
```

```
2
DEL success
```
结果正确。

5.4 test4_0.py、test4_1.py、test4_2.py

test4_0.py 结果：

```
指令序列："PUT A 1","PUT B 1"

PUT success
PUT success
```

test4_1.py 开启 3 个程序循环执行 1 分钟：
进入 test4_1.py 目录下执行：
start cmd /C python test4_1.py
start cmd /C python test4_1.py
start cmd /C python test4_1.py

```
指令序列："BEGIN","PUT A (A+1)","PUT B (B+1)","PUT A
(A+1)","PUT B (B+1)","PUT A (A+1)","PUT A (A+1)","PUT
B (B+1)","PUT B (B+1)","PUT A (A+1)","PUT B
(B+1)","COMMIT","BEGIN","PUT A (A+1)","PUT B
(B+1)","PUT A (A+1)","PUT B (B+1)","PUT A (A+1)","PUT
A (A+1)","PUT B (B+1)","PUT B (B+1)","PUT A (A+1)","PUT
B (B+1)","COMMIT","BEGIN","PUT A (A-1)","PUT B
(B-1)","PUT A (A-1)","PUT B (B-1)","PUT A (A-1)","PUT
A (A-1)","PUT B (B-1)","PUT B (B-1)","PUT A (A-1)","PUT
B (B-1)","ABORT"
```

test4_2.py 结果：

```
指令序列："GET A","GET B"

1171
1171
```

经过测试，3 个进程同时运行 A、B 能达到 1171，单个进程运行只有 536，速度提高一倍。但该吞吐量距离真实场景的需求仍然有较大差距，瓶颈受 IO、MVCC 副本与原本之间的拷贝、stringstream、落盘等影响。