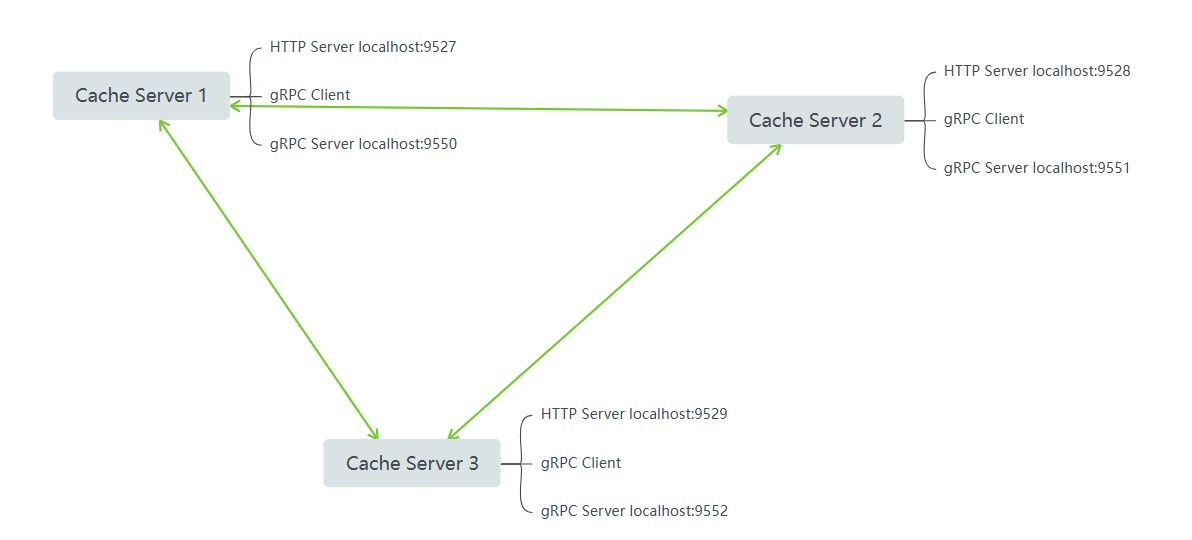
**一、总体实现**

SDSC（Simple Distributed Cache System），简易分布式缓存系统的设计包含三个节点，

每个节点包含三个部分：HTTP Server，职责是对测试脚本中的HTTP请求做出响应、gRPCServer，职责是对其他节点的请求做出响应、gRPCClient，职责是向其他节点发出请求。

 运用了课堂上学到的复制的概念，**在三个节点上保存相同数据的一个副本**。采用了顺序一致性，操作的交错执行序列符合对象单个正确副本所遵循的规约，操作在交错执行中的次序和每个客户程序中执行的次序一致。系统造成了单个副本的错觉，无论哪个客户，读都会返回最近写的结果，无论哪个客户，所有后续读都应返回相同的结果，直到下一次写。在节点之间的数据复制使用了gRPC实现。

**二、HTTP接口实现**

测试脚本中存在不同的HTTP请求，这里的HTTP接口需要根据不同的HTTP请求去调用对应的处理逻辑函数：

func handleHttpRequest(w http.ResponseWriter, r \*http.Request) {

    if r.Method == http.MethodGet {

        handleGet(w, r.URL.String()[1:])

    } else if r.Method == http.MethodPost {

        body, err := ioutil.ReadAll(r.Body)

        if err != nil {

            http.Error(w, "Unable to read request body.", http.StatusInternalServerError)

            return

        }

        handleSet(w, string(body))

    } else if r.Method == http.MethodDelete {

        handleDelete(w, r.URL.String()[1:])

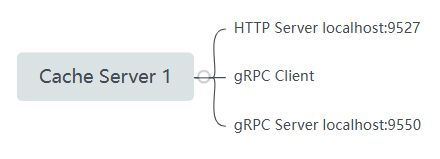
    } else {

        http.Error(w, "Unsupport http request.", http.StatusMethodNotAllowed)

    }

}

1. **RPC实现**



如图中所表示的，一个节点Cache Server中包含一个gRPC Client 和gRPC Server，从测试脚本过来的HTTP请求进入到此SDSC系统中时，会调用gRPC Client向其他所有节点的gRPC Server发起RPC，不同的HTTP请求所对应的不同的接口调用情况如下所示：

SET请求：此SDSC的设计理念中用到了复制的概念，故而，在顺序一致性思想的指导下，以一致性的思想为实现目标，数据在保存到本地的同时，从本节点的gRPC Client 向其他所有节点的gRPC Server中发起请求，将数据保存到其他节点中。RPC的参数来自于测试HTTP请求自带的键值对。

GET请求：由于SET操作而使得三个节点保有的一致性，故而只需在本地GET，而无需在其他节点中GET。

DELETE请求：正是因为在SET时设下的一致性，所以在执行DELETE请求的时候也应该不破坏这种一致性，从而需要在删除本地数据的同时，也对其他所有节点的数据通过RPC的形式同步DELETE操作。RPC参数同样来自于测试HTTP请求自带的Key，返回值是删除的键值对的数量。

1. **Docker打包**

Dockerfile 构建容器的步骤：

1、复制文件到ubuntu20.04镜像，

2、安装protobuf和go编译器

3、安装go的grpc插件

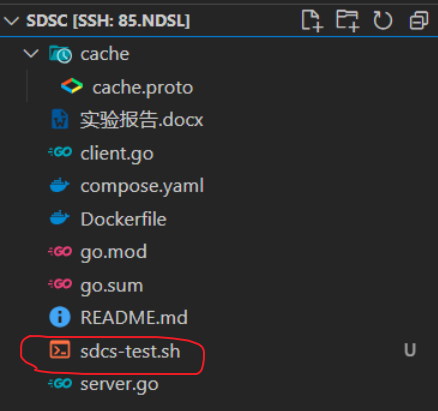
4、编译proto和go源文件

compose.ymal:

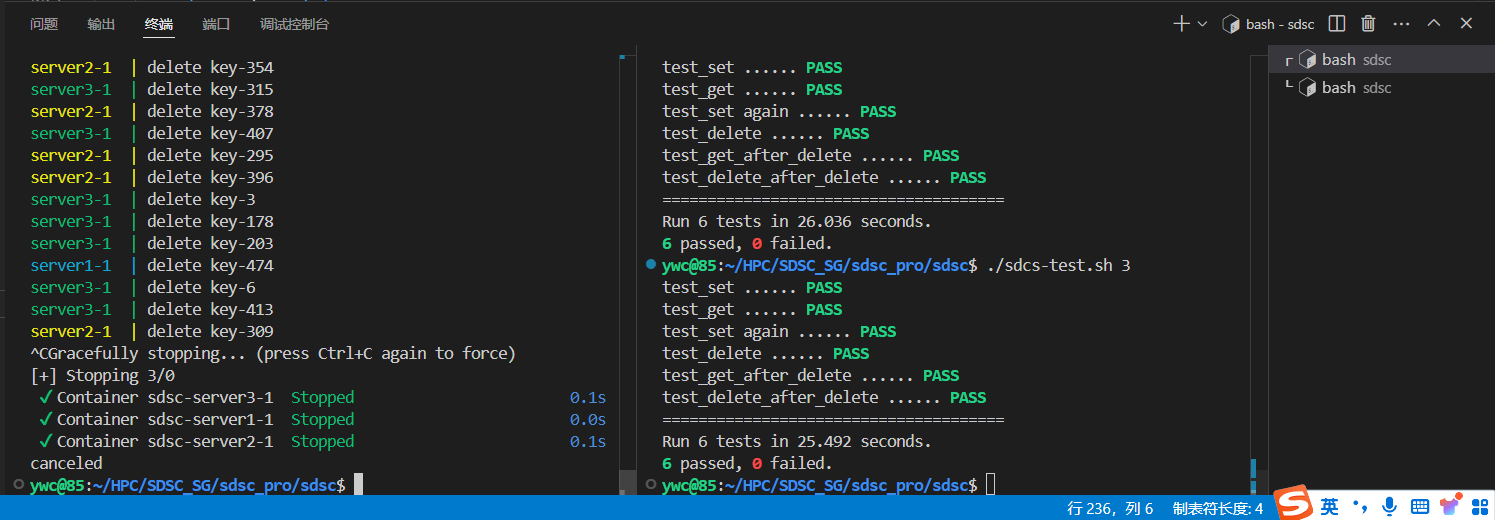
设置镜像和主机共享网络，并设置镜像入口程序为编译的go程序。

**五、实验结果**

测试脚本放置于项目根目录下



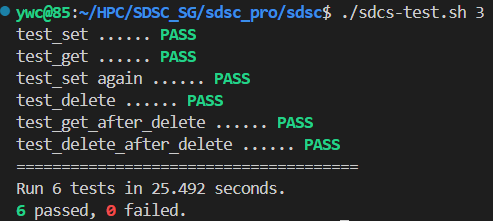
建议测试步骤，建议使用vscode



开两个bash

先在左边输入docker compose up，构建并启动容器，启动服务

然后在右边bash之中输入./sdcs-test.sh 3开始测试



测试结果：6pass 0failed.

Ps:如果将此项目导入vscode中，建议下一个插件



安装即可打开此docx文件。

如：

