

## C4 中的 DP 并网详解

DP=Dispatch 服务，简单来说，DP 的作用就是让 C4 的各个服务节点实现完全互通，而不是让我们每个服务都要指定 ip，端口，对应服务。简单来说，DP 是 C4 的内置的服务桥。

## 在详细讲解 DP 前，我们来回顾一下 C4 的服务器机制

C4 可以做到一个 IP+端口，带起 100 个服务，这些服务由 API: **RegisterC40** 来完成  
在 C4 主库中 RegisterC40 会使用标识符注册一堆双通道内置服务

```
// build-in registration
RegisterC40('DP', TC40_Dispatch_Service, TC40_Dispatch_Client);
RegisterC40('NA', TC40_Base_NoAuth_Service, TC40_Base_NoAuth_Client);
RegisterC40('DNA', TC40_Base_DataStoreNoAuth_Service, TC40_Base_DataStoreNoAuth_Client);
RegisterC40('VA', TC40_Base_VirtualAuth_Service, TC40_Base_VirtualAuth_Client);
RegisterC40('DVA', TC40_Base_DataStoreVirtualAuth_Service, TC40_Base_DataStoreVirtualAuth_Client);
RegisterC40('D', TC40_Base_Service, TC40_Base_Client);
RegisterC40('DD', TC40_Base_DataStore_Service, TC40_Base_DataStore_Client);
```

在 C4 的 FS 库中，RegisterC40 会注册 FS 作为标识符

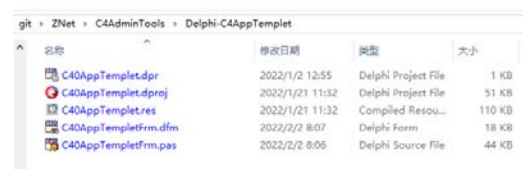
```
Initialization
RegisterC40('FS', TC40_FS_Service, TC40_FS_Client);
end.
```

在 C4 的 UserDB 库中，RegisterC40 会注册 UserDB 作为标识符

```
RegisterC40('UserDB', TC40_UserDB_Service, TC40_UserDB_Client);
```

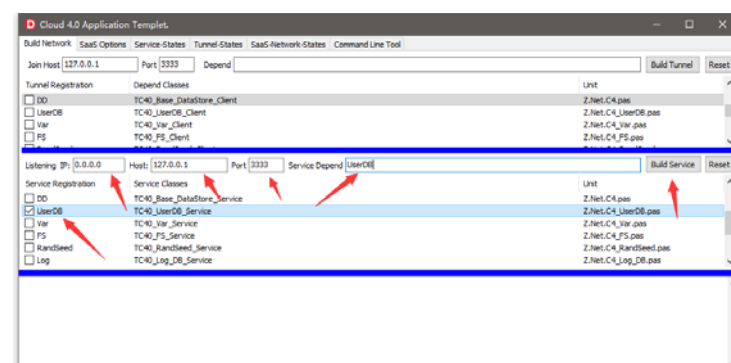
## 使用 C4AppTemplet 开启标识符

C4AppTemplet 分为 FPC/Delphi 两个版本，在目录 C4AdminTool 可以找到它们



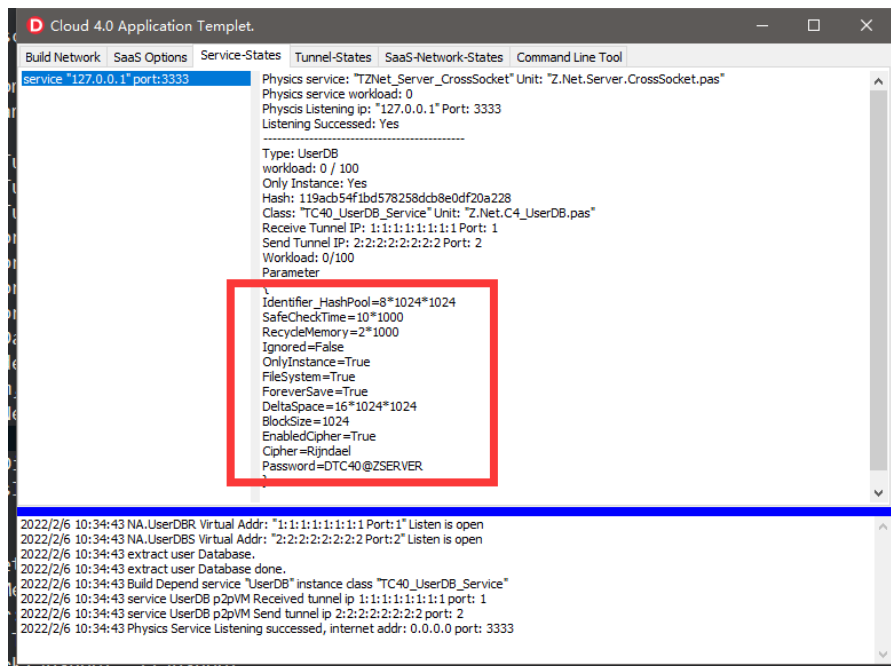
编译启动以后，所有注册过的标识符都会罗列出来，窗口下面是服务，上面的客户端，也就是使用这些服务器的 api。

下图为开启 UserDB 服务，选中标识符，给出正确的 ip，端口，然后 Build Service，这时候 UserDB 就已经开启完成了



## 在 C4AppTemplet 中查看 UserDB 的附加启动参数

必须先 Build Service, 之后, 在 Service-states 页面可以找到 UserDB 的启动参数



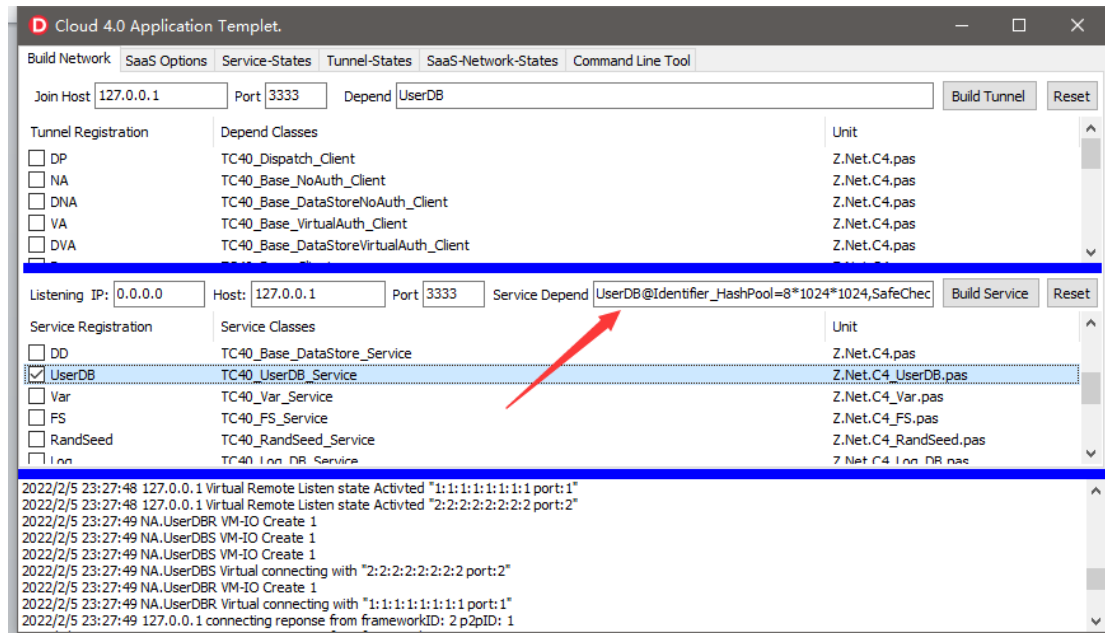
- **SafeCheckTime=60000:** C4 服务使用的数据都有 IO 同步功能, 简单来说, 就是数据在使用时才会存在于内存, 不用的时候都会通过 IO 同步到磁盘保存。这一技术体系为 ZDB2。在 C4 中, SafeCheckTime 表示在 60000 毫秒, 也就是 1 分钟以后, 临时内存会全部同步到磁盘, 并且给服务器释放内存
- **Ignored=False:** DP 同步网络时是否这个服务节点信息, 如果 Ignore 给 True,
- **OnlyInstance=True:** UserDB 标识只允许是 C4 网络的唯一服务, 为 True 时, C4 网络开两个 UserDB 服务将会无法并网
- **FileSystem=True:** 双通道内置的文件系统, 这个开关为单独服务器开关, 全局开关可以修改 Z.Define.inc 中的 DoubleIOFileSystem 定义
- **ForeverSave=True:** 用户数据会永久存储, 如果给 False 服务器在启动时会重置用户数据库, 在 IOT 这类易断电设备要关闭它, 在云服务器全部打开
- **Identifier\_HashPool=4\*1024\*1024:** UserDB 的 Hash 表大小, 在 64 位系统中, UserDB 可带上亿个注册用户, Hash 越大, 根据 Username 查询 UserDB 的速度越快
- **RecycleMemory=60\*1000:** 与 SafeCheckTime 机制类似, 如果用户信息在 1 分钟后没有被访问, 那么用户信息将会写入磁盘 IO 并释放内存
- **DeltaSpace=16\*1024\*1024:** ZDB2 的磁盘空间系统机制, 表示当 UserDB 数据存储空间不够用时, 自动扩容的步进大小, 这里以表达式给的 16M, 既: 空间占满以后, 每次增容 16M 磁盘空间
- **BlockSize=1024:** ZDB2 的磁盘空间系统机制, 每个用户信息初始只占用 1K 的存储块, 简单来说, 如果用户信息体积为 1025, 那么将会使用两个 Block 来存, 既 2k 磁盘空间
- **EnabledCipher=True:** ZDB2 的磁盘空间系统机制, 加密开关
- **Cipher=Rijndael:** ZDB2 的磁盘空间系统机制, 加密算法
- **Password=DTC40@ZSERVER:** ZDB2 的磁盘空间系统机制, 密钥

## 在 C4AppTemplet 中修改 UserDB 的附加启动参数

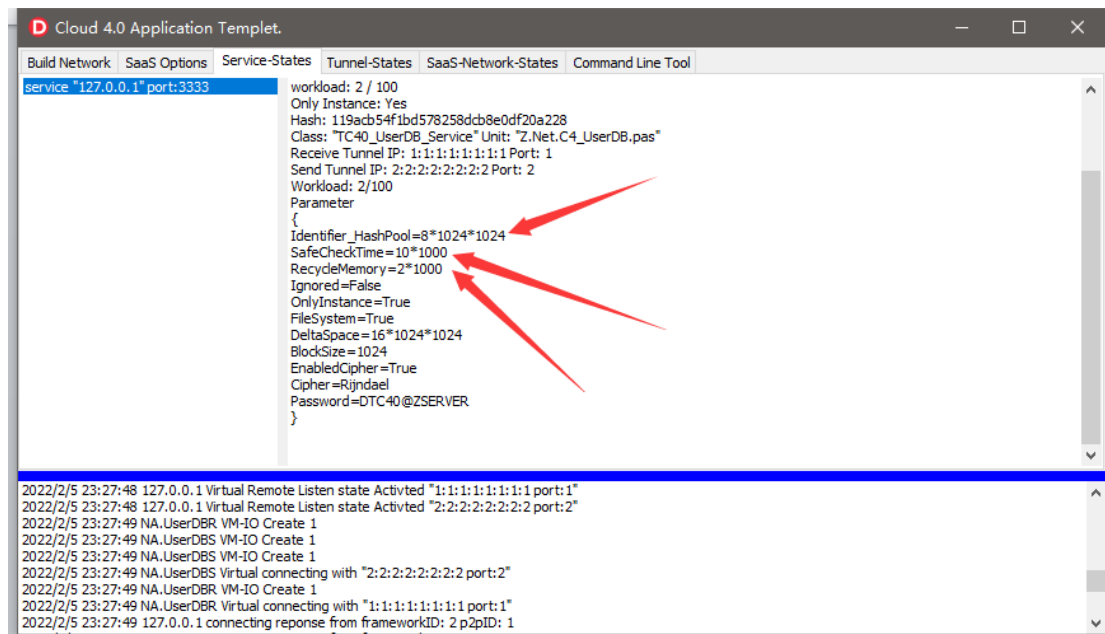
先用工具查看附加启动参数，然后再按 格式，标识符@参数，参数，参数，给出来

Depend `UserDB@Identifier_HashPool=8*1024*1024,SafeCheckTime=10*1000,RecycleMemory=2*1000`

启动参数可以直接按格式写在 Service Depend 栏目中



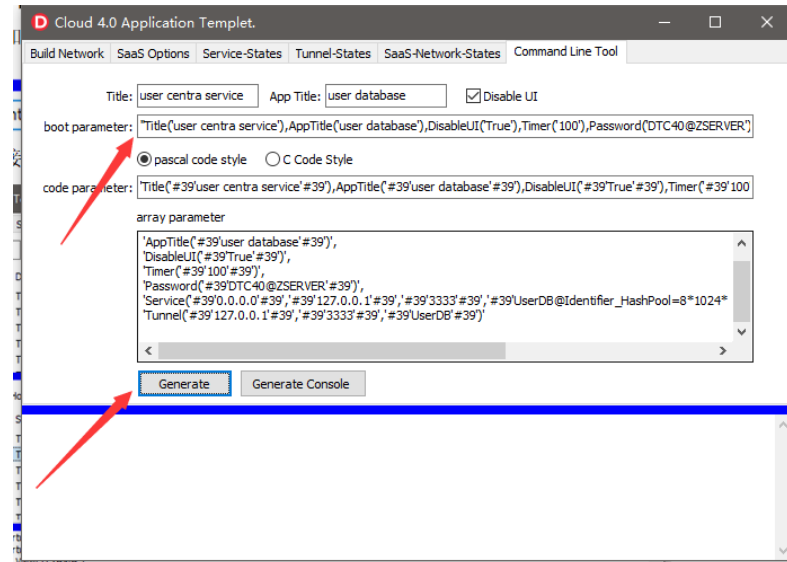
完成后再次查看 UserDB 的启动参数



## 用 C4AppTemplet 生成自动化命令行

自动命令行是 **C4AppTemplet** 内部实现的启动脚本机制，通过命令行生成 UserDB 自动化启动命令可以很方便的部署 C4 到服务中

在非 windows 系统, 自动化命令行可以使用 **Z.Net.C4\_Console\_APP** 库来代替 **C4AppTemplet**, 例如 Linux, IOT 这类设备



生成后的命令行如下（命令太长我在文档剪短了）

```
"Password('DTC40@ZSERVER'),Service('0.0.0.0','127.0.0.1','3333','UserDB')"
```

之后，用命令行方式启动程序即可实现自动化启动服务

```
C4AppTemplet.exe "Password('DTC40@ZSERVER'),Service('0.0.0.0','127.0.0.1','3333','UserDB')"
```

## C4AppTemplet 的 console 版本

Console 版本可以兼容 C4AppTemplet 的命令行脚本，并且可以工作于 console 版本的 windows 和 linux 服务器系统，这里做一个简单的技术介绍

首先在 C4AppTemplet 把系统跑起来，然后生成 console 的启动脚本

C4 的 Delphi Console 范例工程在 ZNet 项目目录中

> git > ZNet > C4AdminTools > Delphi-C4ConsoleTemplet				
名称	修改日期	类型	大小	
C4ConsoleTemplet.dpr	2022/2/6 9:19	Delphi Project File	1 KB	
C4ConsoleTemplet.dproj	2022/2/6 9:19	Delphi Project File	62 KB	
C4ConsoleTemplet.dproj.local	2022/2/6 9:19	LOCAL 文件	3 KB	
C4ConsoleTemplet.identcache	2022/2/6 9:19	IDENTCACHE 文件	1 KB	
C4ConsoleTemplet.res	2022/2/6 9:19	Compiled Resou...	1 KB	

## FPC-C4AppTemplet 的 Console 版本（支持 linux server/osx server）

ZNet 项目可以脱离 FPC 的 LCL 库，不依赖 LCL 对于 FPC 程序来说是 NoUI 的程序运行方式。

由于不依赖 LCL，ZNet 方式可以直接在 Linux Server 系统以自动启动的后台服务运行，不会依赖 gtk 这类图形桌面库

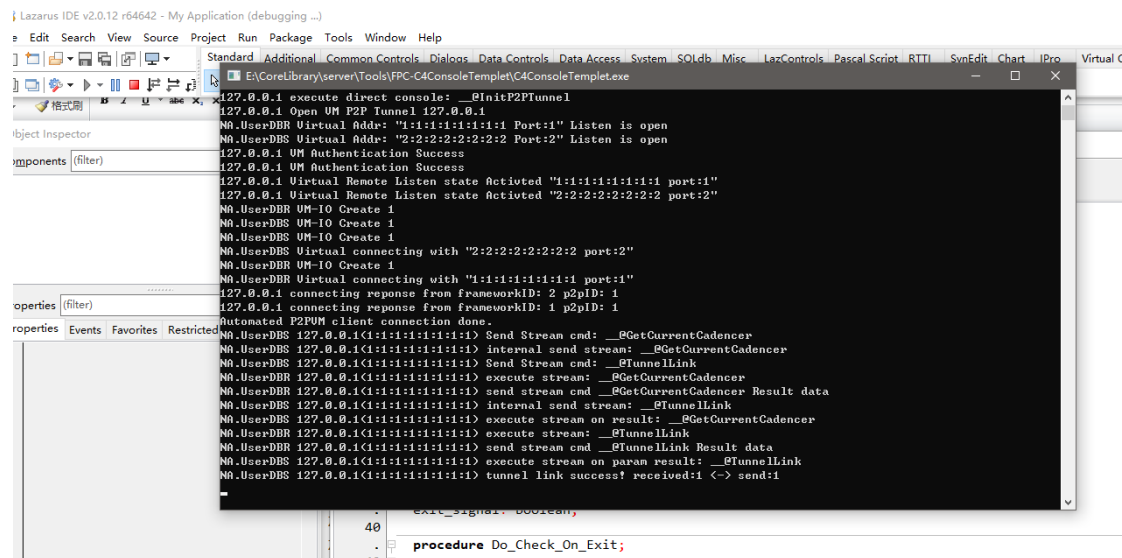
Fpc 的可执行范例工程在 ZNet 项目目录中

名称	修改日期	类型	大小
C4ConsoleTemplet.lpi	2022/2/6 10:18	Lazarus Project I...	2 KB
C4ConsoleTemplet.lpr	2022/2/6 10:18	Lazarus Project ...	2 KB
C4ConsoleTemplet.lps	2022/2/6 10:18	LPS 文件	2 KB
jemalloc_A32.dll	2021/6/16 3:26	应用程序扩展	135 KB
jemalloc_X64.dll	2021/6/16 3:26	应用程序扩展	170 KB

ZNet 可以通过 fpc 构建出工作于基于 Linux 的 IOT/NBIOT 这类移动设备程序




特别注意：ZNet 中带有数据库性质的服务，例如，UserDB，如果拿到 IOT 设备跑服务，断电可能会造成无法修复的数据故障，这时候可以给个参数 ForeverSave=False 这样 IOT 每次在启动时都会重置数据库，从而保证设备正常运行。IOT 设备最好只跑点非数据库的服务，例如 VM，计算等等服务，尽量避免使用任何数据库，如果非要使用数据库，就 ForeverSave=False 以这种方式使用。

当 C4 服务开启以后，不要直接关闭，可以在 console 窗口输入 exit 回车进行正常关闭



```
127.0.0.1 execute direct console: __finitP2PTunnel
127.0.0.1 Open UM P2P Tunnel 127.0.0.1
NA.UserDBR Virtual Addr: "1:1:1:1:1:1:1:1 Port:1" Listen is open
NA.UserDBS Virtual Addr: "2:2:2:2:2:2:2:2 Port:2" Listen is open
127.0.0.1 UM Authentication Success
127.0.0.1 UM Authentication Success
127.0.0.1 Virtual Remote Listen state Activated "1:1:1:1:1:1:1:1 port:1"
127.0.0.1 Virtual Remote Listen state Activated "2:2:2:2:2:2:2:2 port:2"
NA.UserDBR UM-IO Create 1
NA.UserDBS UM-IO Create 1
NA.UserDBS Virtual connecting with "2:2:2:2:2:2:2:2 port:2"
NA.UserDBR UM-IO Create 1
NA.UserDBR Virtual connecting with "1:1:1:1:1:1:1:1 port:1"
127.0.0.1 connecting response from FrameworkID: 2 p2pID: 1
127.0.0.1 connecting response from FrameworkID: 1 p2pID: 1
Automated P2PUM client connection done.
NA.UserDBS 127.0.0.1(1:1:1:1:1:1:1:1) Send Stream cmd: __GetCurrentCadencer
NA.UserDBS 127.0.0.1(1:1:1:1:1:1:1:1) internal send stream: __GetCurrentCadencer
NA.UserDBS 127.0.0.1(1:1:1:1:1:1:1:1) Send Stream cmd: __TunnelLink
NA.UserDBR 127.0.0.1(1:1:1:1:1:1:1:1) execute stream: __GetCurrentCadencer
NA.UserDBR 127.0.0.1(1:1:1:1:1:1:1:1) send stream cmd __GetCurrentCadencer Result data
NA.UserDBS 127.0.0.1(1:1:1:1:1:1:1:1) internal send stream: __TunnelLink
NA.UserDBS 127.0.0.1(1:1:1:1:1:1:1:1) execute stream on result: __GetCurrentCadencer
NA.UserDBR 127.0.0.1(1:1:1:1:1:1:1:1) execute stream: __TunnelLink
NA.UserDBR 127.0.0.1(1:1:1:1:1:1:1:1) send stream cmd __TunnelLink Result data
NA.UserDBS 127.0.0.1(1:1:1:1:1:1:1:1) execute stream on param result: __TunnelLink
NA.UserDBS 127.0.0.1(1:1:1:1:1:1:1:1) tunnel link success! received:1 <-> send:1
EXIT_SIGNAL: 00000000;
48
procedure Do_Check_On_Exit;
49
```

第一，使用 UserDB 前必须明白：**UserDB 是为服务器而设计的用户系统模型，我们必须自己写服务器，而用户的身份验证，大规模存储，IM 消息系统等等，都可以交给 UserDB。这不是让前端使用，而是给服务器用的 UserDB 模型。**

名称	修改日期	类型	大小
 Client	2022/2/2 22:29	文件夹	
 Sever(Linux)	2022/2/2 22:29	文件夹	
 virtualAuthDemo.groupproj	2018/9/16 11:52	Embarcadero D...	2 KB

第一步是使用 VirtualAuth 技术体系构建出 C4 的用户登录身份程序

```

30 | TMyCustom_Service = class(TC40_Base_VirtualAuth_Service)
31 | public
    |     constructor Create(PhysicsService_: TC40_PhysicsService; ServiceTyp, Param_: U_String); override;
    |     destructor Destroy; override;
    |     end;
    |
    | TMyCustom_Client = class(TC40_Base_VirtualAuth_Client)
    | public
        |     constructor Create(source_: TC40_Info; Param_: U_String); override;
        |     destructor Destroy; override;
40 |     end;

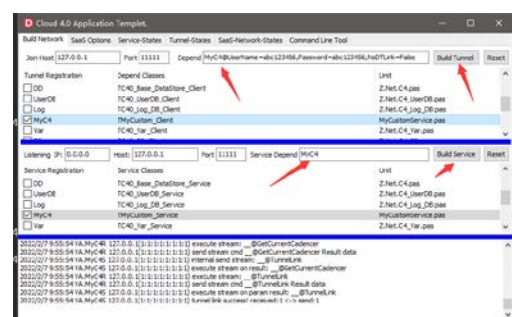
```

Examples

名称	修改日期	类型	大小
50	2024/2/7 22:29	文件夹	
51, OrderStructureThread	2024/2/7 22:29	文件夹	
52, SLEET_P2PVM_Demo	2023/1/10 8:40	文件夹	
53, SLEET_P2PVM_Nodutn	2023/1/10 8:40	文件夹	
54, SLEET_P2PVM_Nodutn_Custom	2023/1/10 8:40	文件夹	
54, SLEET_P2PVM_VirtualKube	2023/1/10 8:40	文件夹	
54, SLEET_P2PVM_VirtualKube_Custom	2023/1/10 8:40	文件夹	
56, SLEET_P2PVM	2023/1/10 8:40	文件夹	
56, SLEET_P2PVM_Custom	2023/1/10 8:40	文件夹	
116, RealTimeDataFrameEncoder	2023/2/7 22:29	文件夹	
117, ZOR2_Non_O6	2023/2/7 22:29	文件夹	

```
30 initialization
-
- RegisterC40('MyC4', TMyCustom_Service, TMyCustom_Client);
33
- end.
```

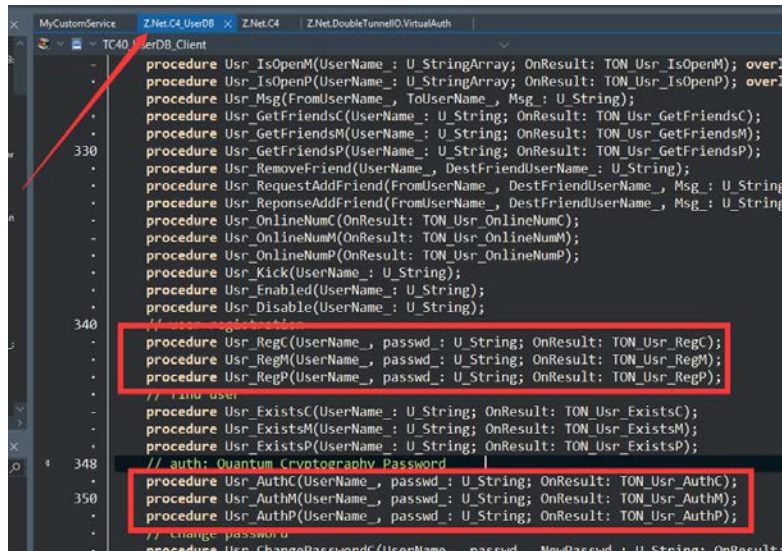
测试 MyC4，可以直接在 C4AppTemplet 里面干，下面是 C4 服务，上面是 C4 客户端





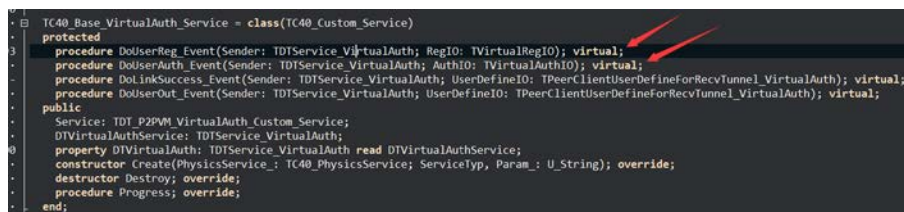
请大家注意: 因为是基于 VirtualAuth 的技术体系做用户登录身份验证, 深入了解 VirtualAuth 是很有必要的。接下来就是接口 UserDB 模型的技术细节了

一, UserDB 对于服务器端的用户注册和身份验证提都供了相应的 api



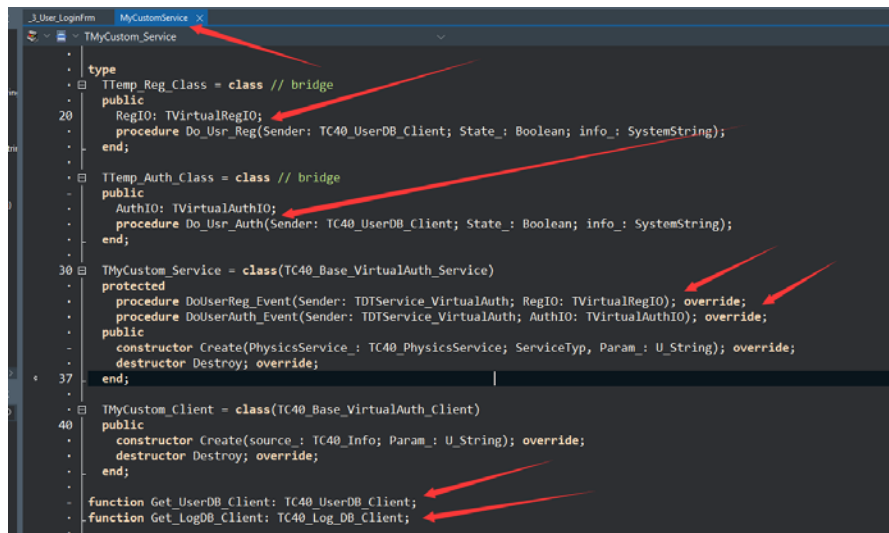
```
procedure Usr_IsOpenM(UserName : U_StringArray; OnResult: TON_Usr_IsOpenM); over
procedure Usr_IsOpenP(UserName : U_StringArray; OnResult: TON_Usr_IsOpenP); over
procedure Usr_Msg(FromUserName, ToUserName, Msg : U_String);
procedure Usr_GetFriendsC(UserName : U_String; OnResult: TON_Usr_GetFriendsC);
procedure Usr_GetFriendsM(UserName : U_String; OnResult: TON_Usr_GetFriendsM);
procedure Usr_GetFriendsP(UserName : U_String; OnResult: TON_Usr_GetFriendsP);
procedure Usr_RemoveFriend(UserName, DestFriendUserName : U_String);
procedure Usr_RequestAddFriend(FromUserName, DestFriendUserName, Msg : U_String);
procedure Usr_ReponseAddFriend(FromUserName, DestFriendUserName, Msg : U_String);
procedure Usr_OnlineNumC(OnResult: TON_Usr_OnlineNumC);
procedure Usr_OnlineNumM(OnResult: TON_Usr_OnlineNumM);
procedure Usr_OnlineNumP(OnResult: TON_Usr_OnlineNumP);
procedure Usr_Kick(UserName : U_String);
procedure Usr_Enabled(UserName : U_String);
procedure Usr_Disable(UserName : U_String);
// user registration
procedure Usr_RegC(UserName, passwd : U_String; OnResult: TON_Usr_RegC);
procedure Usr_RegM(UserName, passwd : U_String; OnResult: TON_Usr_RegM);
procedure Usr_RegP(UserName, passwd : U_String; OnResult: TON_Usr_RegP);
// find user
procedure Usr_ExistsC(UserName : U_String; OnResult: TON_Usr_ExistsC);
procedure Usr_ExistsM(UserName : U_String; OnResult: TON_Usr_ExistsM);
procedure Usr_ExistsP(UserName : U_String; OnResult: TON_Usr_ExistsP);
// auth: Quantum Cryptography Password
procedure Usr_AuthC(UserName, passwd : U_String; OnResult: TON_Usr_AuthC);
procedure Usr_AuthM(UserName, passwd : U_String; OnResult: TON_Usr_AuthM);
procedure Usr_AuthP(UserName, passwd : U_String; OnResult: TON_Usr_AuthP);
// change password
procedure Usr_ChangePasswordC(UserName, passwd, NewPasswd : U_String; OnResult:
```

二, VirtualAuth 技术体系, 通过接口 Reg+Auth 直接访问 UserDB 的 api 就可以了



```
protected
procedure DoUserReg_Event(Sender: TDTService_VirtualAuth; RegIO: TVirtualRegIO; virtual;
procedure DoUserAuth_Event(Sender: TDTService_VirtualAuth; AuthIO: TVirtualAuthIO; virtual;
procedure DoLinkSuccess_Event(Sender: TDTService_VirtualAuth; UserDefineIO: TPeerClientUserDefineForRecvTunnel_VirtualAuth); virtual;
procedure DoUserOut_Event(Sender: TDTService_VirtualAuth; UserDefineIO: TPeerClientUserDefineForRecvTunnel_VirtualAuth); virtual;
public
Service: TDT_P2PVM_VirtualAuth_Custom_Service;
DTVirtualAuthService: TDTService_VirtualAuth;
property DTVirtualAuth: TDTService_VirtualAuth read DTVirtualAuthService;
constructor Create(PhysicsService: TC40_PhysicsService; ServiceTyp, Param : U_String); override;
destructor Destroy; override;
procedure Progress; override;
end;
```

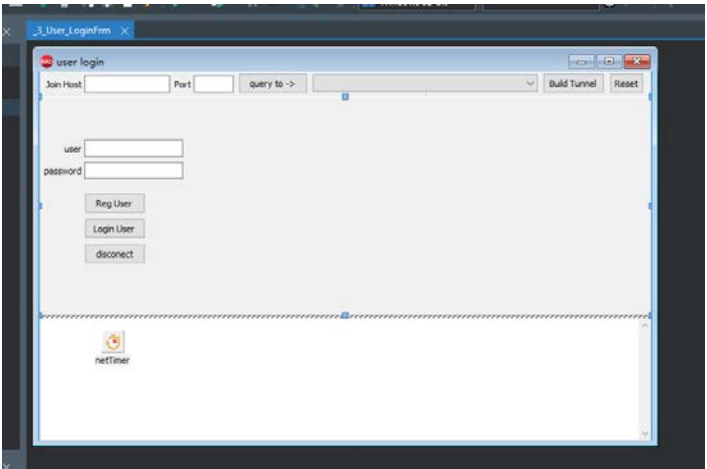
三, 最后, 基于 VirtualAuth 的 TMyCustom\_Service 接口出来就是这样  
可以通过 MyCustomService.pas 代码自己去看流程



```
type
TTemp_Reg_Class = class // bridge
public
RegIO: TVirtualRegIO;
procedure Do_Usr_Reg(Sender: TC40_UserDB_Client; State : Boolean; info : SystemString);
end;
TTemp_Auth_Class = class // bridge
public
AuthIO: TVirtualAuthIO;
procedure Do_Usr_Auth(Sender: TC40_UserDB_Client; State : Boolean; info : SystemString);
end;
TMyCustom_Service = class(TC40_Base_VirtualAuth_Service)
protected
procedure DoUserReg_Event(Sender: TDTService_VirtualAuth; RegIO: TVirtualRegIO); override;
procedure DoUserAuth_Event(Sender: TDTService_VirtualAuth; AuthIO: TVirtualAuthIO); override;
public
constructor Create(PhysicsService : TC40_PhysicsService; ServiceTyp, Param : U_String); override;
destructor Destroy; override;
end;
TMyCustom_Client = class(TC40_Base_VirtualAuth_Client)
public
constructor Create(source : TC40_Info; Param : U_String); override;
destructor Destroy; override;
end;
function Get_UserDB_Client: TC40_UserDB_Client;
function Get_LogDB_Client: TC40_Log_DB_Client;
```

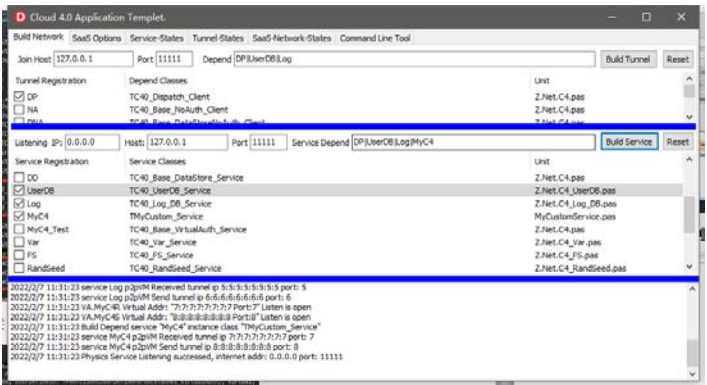
#### 四，做一个前端

前端就是使用 TMyCustom\_Client 的 UI 通讯程序，直接用 D 开堆就行了



#### 部署和测试

C4 的后台部署很灵活，可以把全部服务集成在一个进程，也可以分布式的分开部署



如果分开部署，处于实用性，不建议直接操作 UI 开关服务，写个启动脚本

以下脚本是让所有服务器以 dp 为中心，每次启动时都会往 dp 报告自己的状态，前端登录时，先从 dp 查到可以使用的 MyCustomService，这台 MyCustomService 等同于一台 VM，然后前端只需要挑选其中一个服务做通讯交互就可以了

