# ZNet 使用手冊

文檔版本:1.0
更新時間:2023-10-4

更新日誌:
2023-10-4 開啟撰寫
2023-10-6 完成撰寫

祝大家工作順利,專案大發.

ZNet 相關網站
開源主站 https://github.com/PassByYou888/ZNet
開源備用站 https://gitlab.com/passbyyou888/ZNet
作者個人站 https://zpascal.net

ZNet 作者 qq600585
qq 群, 490269542(開源群),811381795(AI 群)

# 目錄

# 名詞和關鍵機制

- 卡佇列,卡伺服器:卡主迴圈,通訊不流暢,如果伺服器帶有 UI 系統,UI 也會表現出假死
- 阻塞佇列,等待完成,等待佇列:等待是 ZNet 的特有機制,佇列後面會等待前面完成,會嚴格按次序執行,等待會全部在本機等待,只有遠端影響後,本機佇列才會繼續,不是把佇列全部發送過去
- 序列化佇列:不會等待資料回饋,直接發資料,而資料的接收順序是嚴格化的,按 1,2,3 序列發,那麼接收也會是按 1,2,3 進行觸發.例如使用序列化佇列發送 100 條,再發送一條阻塞佇列指令,待阻塞返回既表示 100 條序列已發送完成.同樣,例如上傳一個檔,耗時 1 小時,那麼先發檔,再發條阻塞,待阻塞返回,既表示檔發送成功.
- 非序列化佇列:發送與接收均按嚴格序列機制處理,但是觸發接收後 ZNet 會在某些子執行緒或協程中做解碼這類程式處理,按 1,2,3 序列發送,接收資料以後會放到執行緒中處理,不會按 1,2,3 嚴格序列觸發接收事件.非序列化常用於對資料前後無要求的通訊.

# ZNet 的 6 種命令收發模型

- SendConsole:支援加密,支援壓縮,阻塞佇列模型,每次發送後都會等回饋,例如,先發 100 條命令,最後發一條 SendConsole,回饋回來時也表示 100 條命令都已經發送成功.SendConsole 很輕量,適合收發低於 64K 的小文本.例如 json,xml,ini,yaml.
- SendStream:支援加密,支援壓縮,阻塞佇列模型,每次發送後都會等回饋,所有收發資料都會以 TDFE 進行編碼解碼,由於 TDFE 具備資料容器能力,因此 SendStream 常被用於應用資料收發,SendStream 同樣具備阻塞佇列能力.在流量方面可以支援更大的資料.
- SendDirectConsole:支援加密,支援壓縮,序列化佇列模型,不會等回饋,用於收發基於字串的序列化資料.
- SendDirectStream:支援加密,支援壓縮,序列化佇列模型,不會等回饋,使用容器打包資料,可以收發更大的序列化資料.
- SendBigStream:不支援加密和壓縮,序列化佇列模型,解決超大 Stream 收發,例如檔,超大區塊資料.SendBigStream 工作機制每次只發送一部分,一直等待信號出現才會繼續發送,不會擠爆 socket 緩衝區.物理網路的頻寬和延遲都會影響 SendBigStream 工作效率.
- **SendCompleteBuffer**:不支援加密和壓縮,序列化佇列模型,不會等回饋,高速收發核心機制,**萬兆乙太網支援的核心機制**.CompleteBuffer 設計思路就是圍繞網路來複製 Buffer,高速收發是一個非常重要的機制,取名叫做 CompleteBuffer.

# ZNet 雙通道模型

雙通道是設計層面的概念,表示接收和發送各是一個獨立通道連接,信號收發被區別設計,早期雙通道是創建兩個連接來工作.候來經歷了無數摸索和升級,現在的雙通道是建立在 p2pVM 基礎上,p2pVM 可以在單連接基礎上虛擬出無限多虛擬化連接,這些 p2pVM 連接在應用層都是雙通道.

試想一下,過去我們堆出多台伺服器,需要定義無數多的偵聽,連接,埠,現在用 p2pVM 來虛擬化一切連接.因為降低了後臺技術複雜性,這給後臺系統提供了堆大的可維護性和規範性空間.例如 C4 的所有服務全部走 p2pVM 雙通道.

# ZNet 執行緒支持

ZNet 天生支援在執行緒中發送資料,甚至是並行程式,發送的資料將會是非序列化佇列.

ZNet 的資料接收環節總是位於 Progress 焦點中,也就是觸發 Progress 那個執行緒,在多數情況下,ZNet 都建議在主執行緒中執行 Progress 主迴圈,例如 C4 框架就是用主執行緒跑 Progress.

ZNet 可以支持在執行緒中跑 Progress 從而達到執行緒收發,但並不建議這樣幹,因為 ZNet 有更好執行緒分載方案.

## HPC Compute 執行緒分載方案

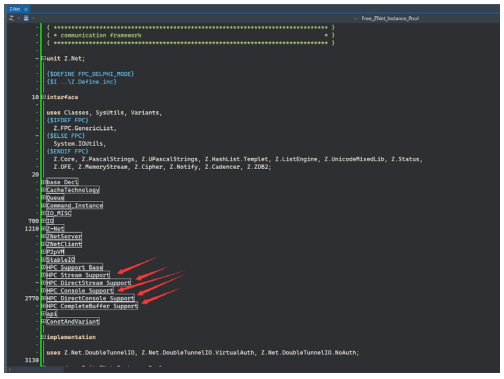
在 ZNet 中以 HPC 開頭的折疊代碼就是執行緒分載解決方案

流程:**當資料從主執行緒到達,觸發接收事件,啟動分載,這時資料會轉移,並且建立一個新執行緒來執行處理,這樣幹以後,伺服器可以運行於無卡頓狀態.因為保護了主執行緒邏輯,穩定性優於自己開 Critical 管理執行緒**.

分載方案可以支援除 BigStream 之外的全部命令收發模型.

分載不可以重疊執行:命令->HPC->HPC->返回,只能:命令->HPC->返回

命令->HPC 被觸發時是走的資料從主執行緒轉移到執行緒,並沒有發生資料 copy



# 萬兆乙太網支援

ZNet 使用 CompleteBuffer 機制來支援萬兆乙太網

- **在 ZNet 內部 SendCompleteBuffer 可以工作與執行緒中**:執行緒機制可以為大流量資料提供預處理這類先決處理條件,例如 10 條執行緒做資料生成,然後 SendCompleteBuffer
- **SendCompleteBuffer 具備高流量緩衝能力**,當成規模的調用 SendCompleteBuffer 將被緩衝到暫存檔案,待主迴圈觸發時成片的 CompleteBuffer 才會被發送出去:該機制對長佇列資料提供了緩存機制,但不可以暴力 Send,一般情況下,10 次 SendCompleteBuffer 可以配上一次 SendNull(阻塞佇列),這樣幹會讓整體網路的負載和吞吐保持良好狀態.
- SendCompleteBuffer 可以與 DirectStream 重疊
  例如**伺服器註冊的命令模型是 DirectStream,可以使用 SendCompleteBuffer 來發送**
- SendCompleteBuffer 支援直接發送 TDFE 資料
- SendCompleteBuffer 可以與 Stream 阻塞模型重疊
  **如果伺服器註冊的命令模型是阻塞 Stream,用戶端使用 SendCompleteBuffer 發送時是非阻塞回應模式**,既發送一個 buffer 資料,也會收到一個 buffer 資料,在這一過程中並不會出現阻塞等待.
- 萬兆乙太收發大資料如果不使用 CompleteBuffer,一個 100M 的資料可能會讓發送端先卡 5 秒並出現 UI 假死,發送出去以後,接收端又會卡頓 5 秒出現回應停頓,這是因為大量的資料複製,編碼,壓縮,解壓縮佔用了主執行緒的計算資源導致,這種機制只能處理體積非常小的資料.
- 在萬兆使用 CompleteBuffer 發送一個 100M 資料,從發送到接收,兩邊的處理延遲可以小於 10ms,這樣的伺服器模型永遠都是立即回應.
- 總結 : **執行緒 + 磁片緩存 +SendNull 阻塞 +DirectStream 重疊 +Stream 重疊 = 用 CompleteBuffer 成功解決萬兆乙太問題**

# 使用 ZNet 必須知道的主迴圈三件事

## 第一件事:遍歷並且處理每個 IO 的資料接收流程

ZNet 的物理網路介面大都用獨立執行緒,這個執行緒是無卡的,當主執行緒被完全佔用,例如正在處理 100M 的壓縮+編碼任務,這時候接收執行緒內部仍然處在正常工作中.**子執行緒接收的資料的並不會放在記憶體中:接收程式會根據資料體量來判斷是否用暫存檔案來暫存資料**.

當主迴圈被觸發時,主迴圈會工作與對應的執行緒中,**主迴圈永遠都是單執行緒模型**,程式會從子執行緒中取出已經收到的資料,包括暫存檔案資料,然後進入網路資料粘包處理環節.

**在資料粘包處理環節,ZNet 使用了大量記憶體投影技術來避免記憶體 copy**,這是將記憶體位址映射成 TMemoryStream,在 ZNet 中記憶體投影使用 TMS64,TMem64 這類工具.

**粘包處理系統會含有 cpu 時間消耗度,由於粘包系統裡面包含了序列包和 p2pVM 這類大型子系統,因此 ZNet 給出了 cpu 時間消耗度,該數值達到臨界點,粘包處理系統將會中斷粘包流程,並且在下次主迴圈才會繼續處理**.例如發送了 1 萬條命令,粘包臨界點是 100ms,當達到 100ms 時粘包只處理了 2000 條命令,那麼剩下的 8000 條將會在下次粘包時進行處理.

在 ZNet 的實際運行中粘包流程幾乎沒有記憶體 copy,單執行緒裡面的粘包處理能力每秒會達到數十萬的處理水準.並不需要開闢執行緒或則協程在處理.**做個執行緒加速這類想法,很不實際**,這不僅會加大 ZNet 的內核複雜度,效率提升也會非常一般,只能是給 Newbie 解決了胡亂高速粘包,而正確的高速粘包,只需要使用 SendCompleteBuffer 發資料進來就行了.

## 第二件事:遍歷並且處理每個 IO 的資料發送流程

ZNet 中的所有發送命令最終都會降落到具體每個 IO 裡面

在這些 IO 中,會有個正在等待發送的命令資料佇列,這些佇列資料,有的會存在於記憶體,有的會存在於暫存檔案.

ZNet 會遍歷並發送 IO 中的待發佇列中的嚴格序列化資料,這些資料將會放到物理的 IO 待發緩衝區中,這一層的緩衝區就不是 ZNet 可以控制的了.

如果使用 p2pVM,StableIO 這類虛擬化通訊協定,在遍歷發送時,嚴格序列化資料會直接被重新封裝,然後再放到物理 IO 緩衝區.

發送的全部物理緩衝資料,是被拓撲網路的信號系統所控制,每個含有 tcp 標籤的 ip 包,都需要有一個終端回饋信號(遠端接收端,不是內網拓撲),這個包才能夠被送到,這種回饋信號就是網路延遲.回饋在 ZNet 裡面是發送速度的快慢.

## 第三件事:管理好每次遍歷的 cpu 開銷

ZNet 會記錄每次訓練遍歷 IO 的時間開銷,達到臨界,遍歷 IO 將會中止,下次主迴圈再繼續遍歷.這樣幹,當伺服器達到一定負載以後,遠端回應將會變慢,而伺服器本身則是在單執行緒的主迴圈中走分片負載的技術路線.

# 優化資料傳輸

首先明確優化目標:避免用戶端卡住 UI 以及避免伺服器卡住主迴圈,這需要作為一個專案或產品整體對待.例如伺服器卡主迴圈,那麼回應速度將會出現延遲,前端發個請求過來,等上 5 秒才回應.

當明確目標後,首要工作是分析卡頓瓶頸,大多數 CS 或 web 型專案,可以直觀通過用戶端發送的命令來定位瓶頸,例如 GetDataList 命令,出現 5 秒卡頓,直接定位到伺服器的 GetDataList 回應環節去就行了.但有時候,ZNet 命令輸送量出現空前規模,例如數百台伺服器之間,以及數千個 IOT 網路設備間的通訊,這些命令密密麻麻,這時就需要 ZNet 提供支援資訊

## 分析瓶頸

如果未使用 C4 框架,需要自己把代碼添加到伺服器應用或則控制台去.

輸出每條命令在伺服器的 cpu 耗損

**ZNet_Instance_Pool.Print_Service_CMD_Info;**

輸出伺服器運行狀態統計

**ZNet_Instance_Pool.Print_Service_Statistics_Info;**

如果使用了 C4 框架,可以直接在控制台輸入 **Service_CMD_Info**

**在輸出命令中,會包含所有命令收發的 cpu 耗損,如果某些資訊含有":HPC Thread"字樣,表示這條命令使用了 HPC 執行緒分載**,例如, GetDataList:HPC Thread": time 123078ms

表示 GetDataList 在 HPC 執行緒分載中最長的一次處理運行了 2 分鐘

如果使用了 C4 框架,可以通過 **HPC_Thread_Info** 控制台命令,即時監控執行緒分載

執行緒分載會給每個執行緒賦予一條正在執行的命令資訊,**HPC_Thread_Info** 會輸出 C4 伺服器的全部執行緒狀態.然後,結合系統的任務和資源監視工具,一直守著務器運行,基本都能分析出性能瓶頸.

## 當分析出性能瓶頸以後,接下來的工作就是解決瓶頸

如果伺服器走主執行緒路線,解決瓶頸只有兩方面工作,**首先是考慮執行緒分載用 HPC 函數開執行緒命令.其次是考慮在用戶端使用 SendCompleteBuffer 來替代 SendDirectStream**.

如果伺服器本身就是多執行緒設計路線,這會需要從鎖,結構,流程這些地方下手來搞,如果從整體來優化多執行緒的伺服器,事情很複雜:**你會從問題的傳導分析再到定位問題點,最後調整流程.最後大概率會使用一個演算法來解決問題**.例如 ZNet 作者的監控專案在搜索視頻時總是等待很久,最終作者設計了一個按時間跨度存視頻的加速演算法.

伺服器堆大以後,一個後臺服務也許會到達 10 萬行規模,很多優化工作,也會是 fixed bug 的工作.**這些優化工作會區分,初期,中期,後期,越靠近初期,fixed bug 的頻率越高,到後期,也許整個伺服器後臺都被推翻重構 1-2 次了,這是一個經驗和設計上的問題**.

# 做伺服器總是圍繞硬體程式設計

很多開源項目看似都很簡單易懂,在真實的伺服器專案中,規模也許會比開源項目大上 100 倍.從量變到質變,規模上升 100 倍,就不再是常規技術方案了.

很多人做伺服器是調度資料庫+通訊系統,如果是做 web,伺服器還會包含設計 ui 系統.如果專案規模很小:通訊頻率低+通訊資料量小,這種伺服器怎麼做都沒有問題.

當伺服器規模開始偏大:通訊頻率高+通訊資料量大,將會面臨:**圍繞硬體程式設計**.

**6 核 12 超線和 128 核 256 超線,這在硬體定位上是不一樣的**,它會影響執行緒和伺服器的設計模型,6 核規格硬體幾乎無法開出並行程式模型,那怕一個環節開出並行 for,也許整個伺服器都會受牽連,6 核的只能開出常規執行緒對特別繁重的計算環節跑執行緒分載.當伺服器運行於 128 核平臺時,就沒有 cpu 計算資源的問題了,而是合理安排計算資源,例如,4 執行緒的平行計算,有時候會比使用 40 個執行緒速度更快.最後是系統瓶頸,在不使用三方執行緒支援函式庫情況下,**win 系統的單進程最多只能同時使用 64 個超線**,只有掛載了 TBB 這類庫以後,才能同時使用 256 個超線.全執行緒,部分執行緒,單執行緒,這種伺服器在設計之初定位就已經完全不一樣.伺服器程式的中心是圍繞不同時代的硬體趨勢,最大的道理硬體平臺,框架設計環節是小道,只有明確了圍繞硬體為中心,再來做伺服器設計和程式設計,這才是正確的路線.

再比如 8 張 4T 全閃 sdd 配 192G 記憶體,以及 8 張 16T 的 hdd 配 1TB 記憶體,這種伺服器在資料存儲,緩存系統設計上也是不一樣的,例如**資料規模到 30 億條,空間佔用到 30T,這種規模基本上 192G 記憶體會很吃緊**,但不是問題,仔細優化後也能跑,因為資料要**加速搜索**或**存儲提速**永遠都是用緩存,而當存放裝置的硬體設定**使用 hdd 並且容量達到 100T,記憶體 1TB,這種設計將比 192G 更加需要優化緩存**,流量進來以後有可能光是寫緩存就直接吃掉 0.99TB 記憶體,程式在設計之初就已經定位好了用 10G 來 hash 索引,開各種優化演算法的結構,這 2 種不同硬體設定,在伺服器的系統設計層面,是不一樣的:192G 只需要考慮優化緩存規模,1TB 需要考慮優化緩存規模+防止崩潰,因為 hdd 寫大資料遇到流量>陣列寫入極限後非常容易崩潰,**大陣列的記憶體一旦用完,陣列的 IO 能力也許會下降到原有能力的 5%,與崩潰無差異**,**緩存控制(flush)將會是直接提上前臺的核心機制之一**,這需要從整體上控制住大資料登錄端,陣列寫機制,硬體鎖這些關鍵要素.

最後是 gpu,一旦伺服器碰上 gpu,支援設備從 cpu 到存儲幾乎全都會是高配,這時候,**伺服器在程式設計環節將會徹底脫離古典的單執行緒方式,流程模型將會被流水線模型所取代**,這些流水線會從一個作業系統到另一個作業系統流來流去.這時候 ZNet 程式會全體走 Buffer+執行緒的路線,並且這種模型只是把資料接進來,計算主體是一堆獨立+巨大的計算支援系統.

# 架橋

架橋是一種通訊模式,有點偏設計模式,它能實實在在提升伺服器群的程式設計效率.
架橋只能工作在帶有回饋請求的命令模型中

- SendStream+SendConsole,帶有佇列阻塞機制的請求,會等回應
- SendCompleteBuffer_NoWait_Stream,不會阻塞佇列的請求,不等回應

架橋的工作流程:

- A->發出請求->命令佇列開始等待模型
- B->收到請求->請求進入延遲回饋模型->架橋 C
- C->收到請求->C 回應請求
- B->收到 C 響應->B 響應回 A
- A->收到 B 響應,跨服通訊流程完結

**架橋是用 1-2 行代碼解決繁瑣的伺服器群間資料傳遞流程,**

```
procedure TDemo_Server.cmd_cb_bridge_stream(Sender: TCommandCompleteBuffer_NoWait_Bridge; InData, OutData: TDFE);
var
  bridge_: TCompleteBuffer_Stream_Event_Bridge;
begin
  // 架桥就是事件指向,剩下的让桥自动处理
  bridge_ := TCompleteBuffer_Stream_Event_Bridge.Create(Sender);
  deploy_bridge.DTNoAuth.SendTunnel.SendCompleteBuffer_NoWait_StreamM('cb_hello_world', InData, bridge_.DoStreamEvent);
end;
```

**在 ZNet-C4 框架中,伺服器的種類數十種,它們,架橋技術是以高效方式來享受這些伺服器資源.在 C4 框架中的全部伺服器都支援架橋與被架橋.**

編寫 C4 伺服器時只管按正常的通訊作業程式設計,直接考慮處理 C 端,無限堆砌.待完成後開個應用伺服器,把所有的伺服器以架橋方式全部調度起來使用即可.

# ZNet 橋支持

ZNet 的橋支援就是事件原型,回應式通訊都會有回饋事件,讓事件指向一個已有的自動程式流程,回饋事件觸發時直接自動處理.

- TOnResult_Bridge_Templet:橋回饋事件原型範本
- TProgress_Bridge:主迴圈橋,掛接到 ZNet 的主迴圈後,每次 progress 都會觸發事件,這種模型在早期 ZNet 還沒有解決 hpc 分載執行緒做資料搜索大流程時,主迴圈橋常被用於分片計算,例如 1000 萬的資料搜索在主迴圈幹就是每次 progress,搜索 10 萬條,以此保證伺服器不卡.
- TState_Param_Bridge:以布林狀態回饋的橋
- TCustom_Event_Bridge:半自動化回應式模型橋,需要程式設計的橋,例如訪問 10 台伺服器,待全部訪問完,再一次性回應給請求端.C4 大量使用.
- TStream_Event_Bridge:SendStream 的自動化回應橋
- TConsole_Event_Bridge:SendConsle 的自動化回應橋
- TCustom_CompleteBuffer_Stream_Bridge:半自動化的 CompleteBuffer 回應事件橋
- TCompleteBuffer_Stream_Event_Bridge: CompleteBuffer 的自動化回應橋

# C4 啟動腳本書寫方式

## win shell 命令列方式:

C4.exe **“server(‘0.0.0.0’,’127.0.0.1’,8008,’DP’)” “KeepAlive(’127.0.0.1’,8008,’DP’)”**

## linux shell 命令列方式

./C4 \
**“server(‘0.0.0.0’,’127.0.0.1’,8008,’DP’)”** \
**“KeepAlive(’127.0.0.1’,8008,’DP’)”** \

## 代碼方式

```
C40AppParsingTextStyle := TTextStyle.tsC; //為了方便書寫腳本,使用 C 風格文本運算式
C40_Extract_CmdLine([
'Service("0.0.0.0", "127.0.0.1", 8008, "DP")',
'Client("127.0.0.1", 8008, "DP")'
    ]);
```

# C4 啟動腳本速查

## 函數:KeepAlive(連接 IP, 連接 Port, 註冊用戶端),

參數重載:**KeepAlive(連接 IP, 連接 Port, 註冊用戶端, 過濾負載)**
別名,支持參數重載:KeepAliveClient, KeepAliveCli, KeepAliveTunnel, KeepAliveConnect, KeepAliveConnection, KeepAliveNet, KeepAliveBuild
說明:用戶端連接伺服器,部署型入網連接,如果連接目標不成功會一直嘗試,連接成功後會自動啟動斷線重連模式.**在部署伺服器群時,主要使用 KeepAlive 方式入網**,無論 C4 的構建參數怎麼變化,KeepAlive 會總是反復嘗試不成功的連接,KeepAlive 方式解決了部署伺服器的啟動順序問題,只要在腳本中使用 KeepAlive 入網可以無視伺服器部署順序問題.KeepAlive 不會搜索整個通訊伺服器棧,需要在 C4 網路中部署 DP 服務,KeepAlive 這樣才能跨服入網.簡單解釋:使用 KeepAlive 入網 C4,需要掛載一個 DP 服務.

## 函數:Auto(連接 IP, 連接 Port, 註冊用戶端)

參數重載:**Auto(連接 IP, 連接 Port, 註冊用戶端, 過濾負載)**
別名,支持參數重載:AutoClient, AutoCli, AutoTunnel, AutoConnect, AutoConnection, AutoNet, AutoBuild
說明:用戶端連接伺服器,非部署型的入網機制,入網失敗後無法自動化反復入網,Auto 是自動型入網連接,可以工作於沒有 DP 服務的 C4 網路,適用於在有人操作啟動的伺服器使用,入網一旦成功就會進入斷線重連模式.

## 函數: Client (連接 IP, 連接 Port, 註冊用戶端)

不支持參數重載
別名,支持參數重載:Cli, Tunnel, Connect, Connection, Net, Build
說明:用戶端連接伺服器,非部署型的入網機制,入網失敗後無法自動化反復入網,Client 函數需要 C4 目標 IP 的網路有 DP 服務才能入網.

## 函數: Service (偵聽 IP, 本機 IP, 偵聽 Port, 註冊伺服器)

參數重載: **Service (本機 IP, 偵聽 Port, 註冊伺服器)**
別名,支持參數重載:Server, Serv, Listen, Listening
說明:創建並啟動 C4 伺服器

## 函數: Wait(延遲的毫秒)

別名,支持參數重載:Sleep
說明:啟動延遲,因為 win32 命令列如果不使用 powershell 腳本,處理延遲執行比較麻煩

## 函數:Quiet(Bool)

說明:安靜模式,預設值 False

## 函數:SafeCheckTime(毫秒)

說明:長週期檢查時間,預設值 45*1000

## 函數:PhysicsReconnectionDelayTime(浮點數,單位秒)

說明:C4 入網以後,如果物理連接斷線,重試連接的時間間隔,預設值:5.0

## 函數: UpdateServiceInfoDelayTime (單位毫秒)

說明:DP 調度伺服器的更新頻率,預設值 1000

## 函數: PhysicsServiceTimeout (單位毫秒)

說明:物理伺服器的連接逾時,預設值 15*10000=15 分鐘

## 函數: PhysicsTunnelTimeout (單位毫秒)

說明:物理用戶端的連接逾時,預設值 15*10000=15 分鐘

## 函數: KillIDCFaultTimeout (單位毫秒)

說明:IDC 故障判定,斷線時長判定,達到該值觸發 IDC 故障,斷線的用戶端會被徹底清理掉
預設值 h24*7=7 天

## 函數: Root (字串)

說明:設置 C4 工作根目錄,預設值為.exe 檔目錄,或則 linux execute prop 檔案名錄.

## 函數: Password (字串)

說明:設置 C4 的入網密碼,預設值為 DTC40@ZSERVER

## UI 函數: Title (字串)

說明:只能工作與 C4 的標注 UI 範本,設置 UI 視窗標題

## UI 函數: AppTitle (字串)

說明:只能工作與 C4 的標注 UI 範本,設置 APP 標題

## UI 函數: DisableUI (字串)

說明:只能工作與 C4 的標注 UI 範本,遮罩 UI 操作

## UI 函數: Timer (單位毫秒)

說明:只能工作與 C4 的標注 UI 範本,設置 UI 環境下的主迴圈毫秒週期

# C4 Help 命令

Help 命令是 C4 內置的伺服器維護+開發調試命令.無論是 console 還是 ui,都內置了 help 命令,這些命令是通用的.

## 命令:help

說明:顯示可用命令清單

## 命令:exit

別名:**close**
說明:關閉伺服器

## 命令:service(ip 位址, 埠)

重載參數: **service(ip 地址)**
重載參數: **service()**
別名:**server**,**serv**
說明:伺服器內部資訊報告,包括物理伺服器資訊,p2pVM 伺服器,連接數量,流量,伺服器內置啟動參數.如果空參數會簡易報告.

## 命令:tunnel(ip 位址, 埠)

重載參數: **tunnel(ip 地址)**
重載參數: **tunnel()**
別名:**client**,**cli**
說明:用戶端內部資訊報告.如果空參數會簡易報告.

## 命令:reginfo()

說明:輸出已經註冊的 c4 服務,c4 的每個服務都會有對應的 CS 模組,例如 DP 會有 dp 伺服器+dp 用戶端.

## 命令:KillNet(ip 位址, 埠)

重載參數: **KillNet (ip 地址)**
說明:直接以 IDC 故障方式殺掉對應的 c4 網路服務

## 命令:Quiet(布林)

重載參數: **SetQuiet(布林)**
說明:切換安靜模式,在安靜模式下,伺服器不會輸出日常命令執行狀態,但出錯有提示,例如命令模型執行異常

## 命令:Save_All_C4Service_Config()

說明:

立即保存當前伺服器參數,這是一個伺服器擴展參數,當 c4 堆大以後伺服器參數太多太多,shell 命令列最長限制是 8192,在正常情況下,根本無法在 shell 命令寫太多啟動參數,因此 c4 提供了檔形式的參數載入方式,在預設情況下,並沒有參數檔

通過 **Save_All_C4Service_Config()**可以生成尾碼為.conf 的伺服器參數檔,.conf 檔存放在當前伺服器目錄對應的 depnd 子目錄中,.conf 是個 ini 格式的設定檔.

如果使用.conf 作為伺服器參數來啟動 c4,命令列的參數將會被覆蓋.

**Save_All_C4Service_Config()**多用於首次運行伺服器時部署啟動參數使用,主要是作用是減少命令列的輸入規模.真實系統集成中,命令列達到一個 200,300 字元,這是非常不易於閱讀修改的.因此.conf 啟動參數是部署 c4 的重要環節.

## 命令: Save_All_C4Client_Config()

說明:立即保存當前構建完成的所有 C4 客端參數,作用與 **Save_All_C4Service_Config()**基本一致.在系統集成工作中,用戶端參數都很少,這是可以直接寫進 shell 命令列的,但如果要美化命令列,使其易於閱讀,那就用文件參數把.

## 命令: HPC_Thread_Info()

說明:立即輸出當前進程中的全部 **TCompute** 執行緒實例,Z 系執行緒一律使用 **TCompute** 創建與執行,並且每個執行緒都會有個 **thread_info** 的字串識別字,用於識別這條執行緒的作用.在 ZNet 中執行緒會非常繁多,有 **HPC 分載執行緒**,**CompleteBuffer 後臺解碼編碼執行緒**,**ZDB2 執行緒**.如果直接使 RT 庫自帶的 **TThread**,那麼 **HPC_Thread_Info()**是不會輸出該執行緒狀態的.

該命令多用於伺服器調試,分析性能瓶頸,找 bug 時使用

## 命令:ZNet_Instance_Info()

別名: **ZNet_Info()**
說明:立即輸出 ZNet 的全部 IO 實例,包括物理連接,p2pVM 連接.多用於診斷連接狀態,分析 C4 入網時遇到的問題.

## 命令: Service_CMD_Info()

別名: **Server_CMD_Info()**
說明:立即輸出伺服器中全部命令模型的 cpu 消耗度統計狀態,這些命令會非常多,數百個.多用於在分析性能瓶頸定位用. **Service_CMD_Info()**也會包含發送命令的次數統計,但不包含發送命令的 cpu 消耗度.

## 命令: Client_CMD_Info()

別名: **Cli_CMD_Info()**
說明:立即輸出全部用戶端命令模型的 cpu 消耗度統計,與 **Service_CMD_Info()**格式幾乎相同,因為 C4 是個交互網路,用戶端統計會折射出伺服器的延遲.

## 命令: Service_Statistics_Info()

別名: **Server_Statistics_Info()**
說明:立即輸出全部伺服器的內部統計資訊,包括 IO 的觸發頻率,加密計算頻率,主迴圈頻率,收發的資料量等等關鍵資訊,伺服器會包括物理伺服器+p2pVM 伺服器

## 命令: Client_Statistics_Info()

別名: **Cli_Statistics_Info()**
說明:立即輸出全部用戶端的內部統計資訊, 輸出格式與 **Service_Statistics_Info()**幾乎相同

## 命令: ZDB2_Info()

說明:立即輸出 ZDB2 的資料庫狀態,ZDB2 是一套分層次架構的資料庫系統,目前 ZDB2 已經進步到第三代體系,這裡的 **ZDB2_Info()**也是輸出第三代 ZDB2 體系,在 C4 框架組成的 FS2,FS,這類服務,凡是 2021 年做出的 C4 服務,都是第二代 ZDB2 體系,無法被 **ZDB2_Info()**統一化的輸出狀態. **ZDB2_Info()**輸出三代體系的信息量和設計非常龐大,這裡只能一筆帶過:在第六代監控的資料庫和後臺用 **ZDB2_Info()**看狀態會是一個好辦法.

## 命令: ZDB2_Flush()

說明:將 ZDB2 寫緩存立即刷入物理設備, 使用資訊與 **ZDB2_Info()**都有非常龐大的信息量,這裡只能一筆帶過:在第六代監控的資料後臺調試陣列系統硬體時用的命令,需要結合磁片 緩存監控,記憶體監控,物理 IO 監控一起來使用.作用是分析出陣列系統的 IO 瓶頸.

# 如何推翻使用 ZNet 的專案

如果之前使用 ZS 走物理雙通道的專案,推翻直接換 C4.

如果之前使用非 C4 雙通道,推薦直接換 C4.

如果專案已經是 C4,可以換個註冊名,**RegisterC40('MY_Serv', TMY_Serv, TMY_Cli)**,然後基於 C4 再重新開一個項目 **RegisterC40('MY_Serv2.0', TMY_Serv2, TMY_Cli2)**

# 如何使用 ZNet 開發 web 類專案

資料通訊層直接 ZNet 跑,UI 層用 webapi 訪問 ZNet 的專案即可.例如 Post+Get 基本能覆蓋 90%的 webapi 需求,ZNet 自帶一個 webapi 的 demo 項目.

# ZNet 與 http 和 web

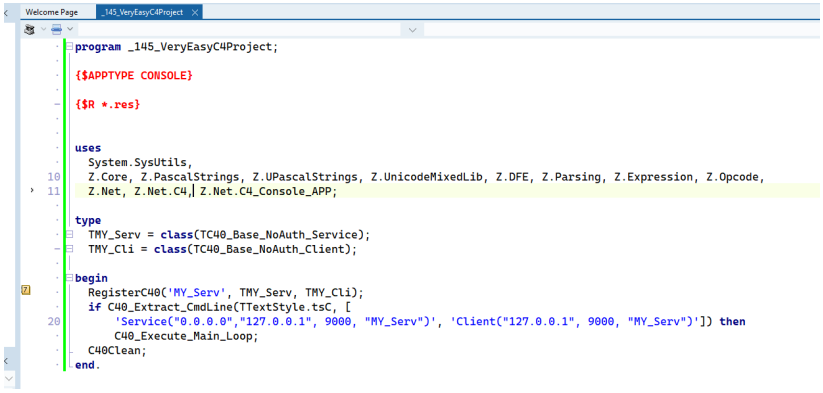
ZNet=大型 CS 伺服器

http=通訊協定

web=全球廣域網路,互聯網

web 包含了 ZNet,在 web 環境下用 apache,nginx 橋通訊模組的大型網站比比皆是,或許讀者有空可以試試用二級功能變數名稱或則域伺服器來做橋接模組和分流,內部如果涉及到大資料或則複雜協定,直接用 ZNet 做個通訊層包 api 給 web 用.

# 文本最後來一個極簡 C4 的 CS demo




全文完.

2023-10-6

by.qq600585