ZNet user manual

Documentation Version :1.1
Updated :2023-10-12

Updated log:
2023-10-4 Start Compose
2023-10-6 Finish writing 1.0
2023-10-9 Added double mainthread and mutex technical data
2023-10-11 Added Performance toolbox information
2023-10-12 Add TCompute concept + structure concept, use the concept to say things because the code can not be told, the code can be mined from large to small.
2023-10-12 appended compilation class technical description, the specific code everyone to dig

I wish you a smooth work, the project.

ZNet related websites
Open source host https://github.com/PassByYou888/ZNet
Open source spare - https://gitlab.com/passbyyou888/ZNet
The author's personal station is https://zpascal.net

ZNet author qq600585
qq group, 490269542(Open source group),811381795(AI group)

Contents

Nouns and key mechanics

- Card queue, card server: card main loop, communication is not smooth, if the server has a UI system, the UI will also show suspended animation
- Block queue, wait for completion, wait queue: Wait is a special mechanism of ZNet, after the queue will wait for the previous completion, will be strictly executed in order, wait will all wait in the local machine, only after remote impact, the local queue will continue, not all the queue sent to the past
- Serialization queue: do not wait for data feedback, directly send data, and the order of data receiving is strict, according to the sequence of 1,2,3, then the receive will also be triggered by 1,2,3. For example, using the serialized queue to send 100, and then send a blocking queue instruction, to block the return indicates that 100 sequences have been sent. Similarly, for example, uploading a file takes 1 hour, then send the file first, and then wind up blocking, and wait for blocking to return, indicating that the file has been successfully sent.
- Non-serialized queue: send and receive are processed according to the strict sequence mechanism, but after triggering the reception, ZNet will do decoding in some sub-threads or coroutines, and send the received data in the sequence of 1,2,3. The received data will be placed in the thread for processing, and the received events will not be triggered according to the strict sequence of 1,2,3. Serialization is often used for communication without any requirement before or after the data.

6 Command sending and receiving models for ZNet

- SendConsole: Supports encryption, compression, blocking queue model, and waits for feedback after each sending. For example, send 100 commands first, and send one SendConsole last. When feedback comes back, it indicates that all 100 commands have been sent successfully. SendConsole is lightweight and suitable for sending and receiving small text less than 64K. Examples are json,xml,ini,yaml.
- SendStream: supports encryption, compression and blocking queue model, and will wait for feedback after each sending. All sent and received data will be encoded and decoded by TDFE. Because TDFE has the capability of data container,SendStream is often used for data sending and receiving, and SendStream also has the capability of blocking queue. In terms of traffic, it can support larger data.
- SendDirectConsole: supports encryption, compression, serialization queue model, does not wait for feedback, used to send and receive serialized data based on string.
- SendDirectStream: supports encryption, supports compression, serializes queue model,

does not wait for feedback, uses container to package data, can send and receive larger serialized data.

- SendBigStream: does not support encryption and compression, serialize the queue model, to solve the problem of sending and receiving large Stream, such as files, large memory block data. SendBigStream only sends a part at a time and waits for the signal to appear before continuing to send, so as not to burst the socket buffer. The bandwidth and latency of the physical network affect SendBigStream.
- **SendCompleteBuffer**: Does not support encryption and compression, serialized queue model, does not wait for feedback, high-speed transceiver core mechanism, **10 Gigabit Ethernet support core mechanism.** CompleteBuffer design idea is to copy Buffer around the network, high-speed sending and receiving is a very important mechanism, named CompleteBuffer.

ZNet dual channel model

Double channel is the concept of the design level, indicating that the receiving and sending are an independent channel connection, the signal sending and receiving are designed differently, the early double channel is to create two connections to work. After countless explorations and upgrades, the current dual channel is built on the basis of p2pVM,p2pVM can be virtualized on the basis of a single connection to an unlimited number of virtualized connections, these p2pVM connections in the application layer are double channels.

Imagine, in the past, we piled multiple servers, need to define countless more intercepts, connections, ports, and now use p2pVM to virtualize all the connections. Because of the reduction of background technical complexity, this provides the background system with a large heap of maintainability and regulatory space. For example, all the services of C4 go through p2pVM dual channels.

ZNet threading support

ZNet naturally supports sending data in threads, even parallel programs, that will be sent to an unserialized queue.

The data receiving part of ZNet is always in the Progress focus, which is the thread that triggers Progress. In most cases,ZNet recommends that the Progress main loop be executed in the main thread, such as the C4 framework that runs Progress in the main thread.

ZNet can support running Progress in a thread to achieve thread sending and receiving, but this is not recommended because ZNet has a better thread load sharing scheme.

HPC Compute thread load sharing scheme

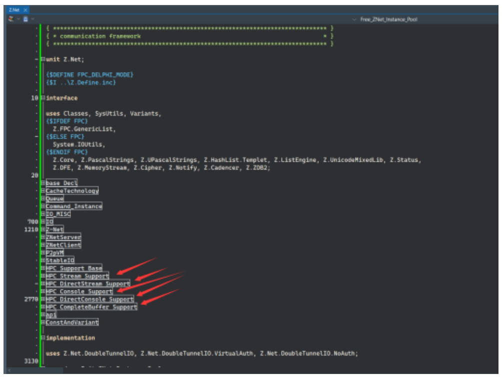
The fold code that starts with HPC in ZNet is the thread load sharing solution

Process: **When the data arrives from the main thread, the receive event is triggered, the load is started, at this time the data is transferred, and a new thread is created to perform the processing, so that the server can run in a stall-free state. Because the main thread logic is protected, it is more stable than running your own Critical management thread**.

The load sharing scheme can support all the command sending and receiving models except BigStream.

The load sharing can not be overlapped: command ->HPC->HPC-> return, only: command ->HPC-> return

Command ->HPC is triggered when the data is transferred from the main thread to the thread, and no data copy occurs



```
{ ****************************************************************************** }
{ * communication framework                                                    * }
{ ****************************************************************************** }

unit Z.Net;

{$DEFINE FPC_DELPHI_MODE}
{$I ..\Z.Define.inc}

interface

uses Classes, SysUtils, Variants,
{$IFDEF FPC}
  Z.FPC.GenericList,
{$ELSE FPC}
  System.IOUtils,
{$ENDIF FPC}
  Z.Core, Z.PascalStrings, Z.UPascalStrings, Z.HashList.Templet, Z.ListEngine, Z.UnicodeMixedLib, Z.Status,
  Z.DFE, Z.MemoryStream, Z.Cipher, Z.Notify, Z.Cadencer, Z.ZDB2;

base Decl
CacheTechnology
Queue
Command_Instance
IO_MISC
IO
Z-Net
ZNetServer
ZNetClient
P2pVM
StableIO
HPC Support Base
HPC Stream Support
HPC DirectStream Support
HPC Console Support
HPC DirectConsole Support
HPC CompleteBuffer Support
api
ConstAndVariant

implementation

uses Z.Net.DoubleTunnelIO, Z.Net.DoubleTunnelIO.VirtualAuth, Z.Net.DoubleTunnelIO.NoAuth;
```

10 Gigabit Ethernet support

ZNet uses the CompleteBuffer mechanism to support 10 Gigabit Ethernet

- **SendCompleteBuffer works with a thread inside ZNet**: The threading mechanism provides pre-processing for large volumes of data, such as 10 threads doing data generation, and SendCompleteBuffer
- **SendCompleteBuffer has a high traffic buffering capacity**. Calls to SendCompleteBuffer as large as possible will be buffered to a temporary file, and chunks of Completebuffers will not be sent out until the main loop is triggered: The mechanism provides caching for long queues of data, but it cannot force Se nd, in general,10 sendCompletebuffers can be configured with one SendNull(blocking queue), which will keep the load and throughput of the overall network in good condition.
- SendCompleteBuffer can overlap with DirectStream
  For example**, the command model registered by the server is DirectStream and can be sent using SendCompleteBuffer**
- SendCompleteBuffer supports sending TDFE data directly
- SendCompleteBuffer can be overlapped with the Stream blocking model
  If the server is registered with a blocking Stream command model, the client **sends completeBuffer** in non-blocking response mode, sending and receiving a buffer without waiting for a block.
- If you do not use the ten-gigabit Ethernet to send and receive big data, a 100M data may make the sender stuck for 5 seconds and the UI will be suspended. After sending, the receiver will be stuck for 5 seconds and the response pause will appear. This is because a large number of data replication, encoding, compression and decompressing occupy the computing resources of the main thread Can handle the volume of very small data.
- Using CompleteBuffer in Ten Gigabit to send a 100M data, from send to receive, the processing delay on both sides can be less than 10ms, such server model is always immediate response.
- Summary: **Thread + Disk cache +SendNull blocking +DirectStream overlap +Stream overlap = Successfully solved Ten Gigabit Ethernet problem with CompleteBuffer**

Three things you must know about the main loop with ZNet

The first thing: go through and process the data receiving process for each IO

ZNet's physical network interface uses mostly independent threads, which are card-free. When the main thread is fully occupied, such as processing a 100M compression + encoding task, the receiving thread is still working internally. The **data received by the child thread is not stored in memory:** the **receiver determines whether to use temporary files to store the data based on the size of the data**.

When the main loop is triggered, the main loop will work with the corresponding thread, the **main loop is always single-threaded model**, the program will remove the data that has been received from the child thread, including temporary file data, and then enter the network data sticky packet processing.

In the process of data sticky packet processing,**ZNet uses a lot of memory projection technology to avoid memory copy**, which maps memory addresses to TMemoryStream. In ZNet, memory projection uses tools like TMS64 and TMem64.

The **sticky packet processing system will contain cpu time consumption**. **Since the sticky packet system contains large-scale subsystems such as** serial packet and **p2pVM**, **ZNet gives** the **cpu time consumption**. When the **value reaches the critical point**, the **sticky packet processing system will interrupt the sticky packet process and continue processing in the next main cycle**. For example, if 10,000 commands are sent, the critical point of sticky packet processing is 100ms, when it reaches 100ms, sticky packet processing only 2000 commands, then the remaining 8000 commands will be processed in the next sticky packet processing.

In the actual operation of ZNet, the sticky packet process has almost no memory copy, and the sticky packet processing capacity in a single thread can reach hundreds of thousands of processing levels per second. No thread or coroutine is needed to process the process. The **idea of doing a thread acceleration is not practical**, it will not only increase the kernel complexity of ZNet, the efficiency increase will be very general, only for Newbie to solve the random high-speed sticky packet, and the correct high-speed sticky packet, only need to use SendCompleteBuffer to send the data in.

The second thing: go through and process the data sending flow of each IO

All send commands in ZNet will eventually land in each IO

In each IO, there will be a queue of command data waiting to be sent. These queue data will either be stored in memory or in temporary files.

ZNet will iterate through and send strictly serialized data from the queue in the IO. This data will be placed in the physical IO buffer, which is not under ZNet's control.

If a virtualization protocol such as p2pVM or StableIO is used, the rigorously serialized data is directly repackaged and placed in the physical IO buffer when traversed.

All the physical buffer data sent is controlled by the signal system of the topological network. Each ip packet containing tcp labels needs to have a terminal feedback signal (remote receiver, not Intranet topology) before the packet can be sent. This feedback signal is the network delay. The feedback in ZNet is the speed of sending.

Third thing: Manage the cpu overhead per traversal

ZNet will keep track of the time spent on each training traversal IO. At a critical point, the traversal IO will be aborted and the main loop will resume the traversal the next time. In this way, when the server reaches a certain load, the remote response will slow down, and the server itself will go through the technical route of fragmentation load in the single-threaded main loop.

Optimize Data transfer

First define the optimization goal: avoid client jamming UI and avoid server jamming main loop, this needs to be treated as a project or product. For example, the server card main loop, then the response speed will be delayed, the front end sends a request, wait 5 seconds before responding.

When the goal is clear, the first job is to analyze the bottleneck. Most CS or web projects can be located intuitively through the command sent by the client to locate the bottleneck, such as the GetDataList command, if there is a 5-second delay, directly locate the server's GetDataList response link on the line. But sometimes,ZNet command throughput is unprecedented, such as hundreds of servers, and thousands of IOT network devices communication, these commands are dense, and ZNet support information is needed

Analyzing Bottlenecks

If you are not using the C4 framework, you need to add your own code to the server application or console.

Output each command at the server's cpu drain

ZNet_Instance_Pool.Print_Service_CMD_Info;

Output server running status statistics

ZNet_Instance_Pool.Print_Service_Statistics_Info;

If you are using the C4 framework, you can type **Service_CMD_Info** directly into the console

The output command contains the cpu consumption of all commands sent and received. If some information contains ": HPC Thread", it indicates that the command uses HPC Thread sharing. For example**,** GetDataList:HPCThread": time 123078ms

Indicates that the longest processing run of GetDataList in the HPC thread load is 2 minutes

If you are using the C4 framework, you can monitor the thread load in real time with the **HPC_Thread_Info** console command

Thread loading assigns each thread a command information that is being executed, and **HPC_Thread_Info outputs** all the thread status of the C4 server. Then, combined with the system's task and resource monitoring tools, keep the server running all the time, and basically can analyze the performance bottleneck.

After the performance bottleneck is analyzed, the next job is to solve the bottleneck

If the server takes the main line route**, there are** only two aspects to solve the bottleneck**, the first is to consider the thread load with the HPC function to open the thread processing command**. **The second is to consider using SendCompleteBuffer instead of SendDirectStream on the client side**.

If the server itself is a multithreaded design route, this will need to start from the lock, structure, flow and so on. If you optimize the server from the whole multithreaded, things are complicated: **you will** start **from the conduction analysis of the problem to the location of the problem**, and **finally adjust the flow**. **Finally, there is a high probability that an algorithm will be used to solve the problem**. For example, the ZNet author's surveillance project was always waiting for a long time when searching for video, and the author eventually designed an accelerated algorithm to save videos by time span.

After the server stack is large, a background service may reach the scale of 100,000 lines, and a lot of optimization work will also be fixed bug work. **These optimization work will distinguish between the early stage, the middle stage and the later stage. The closer to the early stage, the higher the frequency of fixed bugs. In the later stage, perhaps the whole server background has been overthrown and reconstructed 1-2 times, which is a matter of experience and design**.

The server is always programmed around the hardware

Many open source projects seem easy to understand, and in a real server project, the scale may be 100 times larger than the open source project. From the quantitative change to the qualitative change, the scale increases by 100 times, and it is no longer a conventional technical solution.

Many people do the server is a scheduling database + communication system, if it is to do the web, the server will also include the design ui system. If the project scale is small: low communication frequency + small amount of communication data, this server how to do is no problem.

When the server scale begins to be too large: high communication frequency + large communication data, it will face: **programming around hardware**.

6 core **12 super line and 128** core **256 super line**, **which is not the same in hardware positioning**, it will affect the design model of thread and server,6 core specifications hardware can hardly open parallel program model, that is afraid of a link open parallel for, perhaps the whole server will be implicated,6 core can only open conventional threads for particularly heavy computing links run thread load. When the server runs on the 128-core platform, there is no problem of cpu computing resources, but a reasonable arrangement of computing resources, for example,4 threads of parallel computing, sometimes faster than using 40 threads. Finally, the system bottleneck, in the case of not using the three-party thread support library, **a single process in the win** system can only **use 64** hyperlines at the **same time**, only after mounting TBB library, can use 256 hyperlines at the same time. Full thread, partial thread, single thread, this kind of server in the design of the beginning of positioning has been completely different. The center of the server program is around the hardware trend of different times, the biggest reason hardware platform, frame design link is a path, only clear around the hardware as the center, and then to do the server design and programming, this is the right route.

For example, 8 4T full flash sdd with 192G memory, and 8 16T hdd with 1TB memory, this server in the data storage, cache system design is not the same, such as the **data scale to 3 billion**, **space occupation to 30T**, **this scale basically 192G memory will be very tight**, but not a problem, careful optimization can also run, because of the number According to **speed up the search** or **storage speed** is always with the cache, and when the storage device hardware configuration **using hdd and capacity reached 100T**, **memory 1TB**, **this design will be more than 192G** need to optimize the cache, traffic in the future may just write the cache directly eat 0.99TB memory, the program has been positioned at the beginning of the design with 10G to ha sh index, open the structure of various optimization algorithms, these two different hardware configurations, in the system design level of the server, is not the same :192G only need to consider the optimization of cache size,1TB need to consider the optimization of cache size + to prevent crash, because hdd write big data encounter traffic > array write limit is very easy to crash, **large array of memory once used up**, **array** The **IO capacity may be** reduced to **5%** of the original capacity, **no difference with** the crash, cache **control (flush) will be directly put on the front of one of the core**

mechanisms, which requires the overall control of the big data input, array write mechanism, hardware lock these key elements.

Finally, the gpu, once the server meets the gpu, the support device from the cpu to the storage will almost all be highly configured, at this time, the **server will completely break away from the classical single-thread mode in the program design**, the **process model will be replaced by** the pipeline model, these pipelines will flow from one operating system to another operating system. At this time, the ZNet program will all take the route of Buffer+ thread, and this model only puts the data in, the main body of the calculation is a bunch of independent + huge computing support system.

Building Bridges

Bridging is a communication mode that is a bit of a design mode, and it can really improve the programming efficiency of server farms.

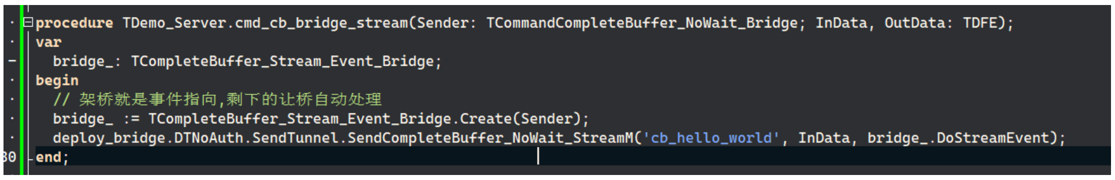
Bridging can only work in a command model with feedback requests

- SendStream+SendConsole, requests with queue blocking, will wait for a response
- SendCompleteBuffer_NoWait_Stream, which does not block requests in the queue, does not wait for responses

Workflow for bridging:

- A-> Make a request -> Command queue starts waiting for the model
- B-> Receive Request -> Request to enter Delayed Feedback Model -> Bridge C
- C-> Request Received ->C Respond to request
- B-> Receive C response ->B respond back to A
- A-> Receive a response from B, the cross-server communication process is complete

Bridging is to solve the cumbersome data transfer process between server farms with 1-2 lines of code.



```
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;
```

In the ZNet-C4 framework, there are dozens of types of servers, they, bridging technology is an efficient way to enjoy these server resources. All the servers in the C4 framework support bridging and bridging.

When writing C4 server, just according to the normal communication operation programming, directly consider dealing with the C end, infinite stacking. After completion, open an application server, and schedule all the servers to bridge the way to use.

ZNet bridge support

ZNet Bridge support is the event prototype, responsive communication will have feedback events, let the event point to an existing automated program flow, feedback events directly automatically processed when triggered.

- TOnResult_Bridge_Templet: Bridge feedback event prototype template
- TProgress_Bridge: main loop bridge, after connecting to the main loop of ZNet, every progress will trigger events. This model in the early ZNet has not solved the hpc load sharing thread to do data search process, the main loop bridge is often used for fragment calculation, for example, 10 million data search in the main loop is every p rogress: Searches for 100,000 packets to ensure that the server does not get stuck.
- TState_Param_Bridge: Bridge with Boolean state feedback
- TCustom_Event_Bridge: semi-automatic responsive bridge. It requires programming, for example, to access 10 servers, and then respond to the requester at one time after all the access is complete. C4 is heavily used.
- TStream_Event_Bridge: An automated response bridge for SendStream
- TConsole_Event_Bridge: Automated response bridge of SendConsle
- TCustom_CompleteBuffer_Stream_Bridge: semi-automated CompleteBuffer response event bridge
- TCompleteBuffer_Stream_Event_Bridge: An automated response bridge for CompleteBuffer

C4 Enable the script writing mode

win shell command line method:

```
C4. Exe "server (127.0.0.1 '0.0.0.0', ', '8008,' DP ')" "KeepAlive (127.0.0.1 ', '8008,' DP ')"
```

linux shell command line

```
./C4 \
"server(' 0.0.0.0 ', '127.0.0.1',8008, 'DP')" \
"KeepAlive(' 127.0.0.1 ',8008, 'DP')" \
```

Code way

C40AppParsingTextStyle := TTextStyle.tsC; // For ease of writing the script, use C-style text expressions

```
C40_Extract_CmdLine([
'Service("0.0.0.0", "127.0.0.1", 8008, "DP")',
'Client("127.0.0.1", 8008, "DP")'
        ]);
```

C4 startup script Quick check

Function :KeepAlive(connect IP, connect Port, register client),

Parameter overload :**KeepAlive(connect IP, connect Port, register client, filter load)**
Aliases, support parameter overloading :KeepAliveClient, KeepAliveCli, KeepAliveTunnel, KeepAliveConnect, KeepAliveConnection, KeepAliveNet, KeepAliveBuild
The client connects to the server, the deployment type of network connection, if the connection target is unsuccessful, it will always try, and the connection will automatically start the disconnection mode after the connection is successful. **In the deployment of server** farms, the **main use** of **KeepAlive mode to enter the network**, no matter how the C4 construction parameters change,KeepAlive will always repeatedly try unsuccessful connections,KeepAlive mode solves the deployment of server startup order, as long as the use of KeepAlive network in the script can ignore the server deployment process The order problem. KeepAlive does not search the entire communication server stack. The DP service needs to be deployed on the C4 network so that KeepAlive can access the network across the server. Simple explanation: To use KeepAlive to connect to C4, you need to mount a DP service.

Function :Auto(Connect IP, connect Port, register client)

Parameter overload :**Auto(Connect IP, connect Port, register client, filter load)**
Aliases, support parameter overloading :AutoClient, AutoCli, AutoTunnel, AutoConnect, AutoConnection,
AutoNet, AutoBuild
Client connection server, non-deployment type access mechanism, access failure can not be automated repeated access to the network,Auto is an automatic access to the network connection, can work in the C4 network without DP service, suitable for use in someone started the server, access to the network once successful will enter disconnection mode.

Function: Client (connect IP, connect Port, register client)

Parameter overloading is not supported
Alias, parameter overloading is supported :Cli, Tunnel, Connect, Connection, Net, Build
The Client connects to the server, the non-deployment type of network access mechanism, the network cannot automatically enter the network repeatedly after the failure, the client function needs the C4 target IP network has DP service to enter the network.

Function: Service (listening IP, local IP, listening Port, registration server)

Parameter overload: **Service (local IP, listening Port, registration server)**
Aliases, support parameter overloading :Server, Serv, Listen, Listening
Description: Create and start the C4 server

Function: Wait(milliseconds of delay)

Alias, support for overloading parameter :Sleep
Description: Startup delay, because win32 command line if not using powershell script, handle the delay execution is more troublesome

Function :Quiet(Bool)

Description: Quiet mode, default is False

Function :SafeCheckTime(ms)

Description: Long cycle check time, the default is 45*1000

Function: PhysicsReconnectionDelayTime (floating point Numbers, the unit seconds)

Note: After C4 is connected to the network, if the physical connection is disconnected, retry the connection interval. Default value :5.0

Function: UpdateServiceInfoDelayTime (ms)

Description :DP schedules the update frequency of the server. The default value is 1000

Function: PhysicsServiceTimeout (in milliseconds)

Note: The connection timeout of the physical server. The default value is 15 x 60 x 1000=15 minutes

Function: PhysicsTunnelTimeout (in milliseconds)

Note: Connection timeout of the physical client. The default value is 15 x 60 x 1000=15 minutes

Function: KillIDCFaultTimeout (in milliseconds)

IDC fault determination, disconnection time determination, reach this value trigger IDC fault, disconnection client will be completely cleaned up
The default value is h24 x 7=7 days

Function: Root (string)

Description: Set the C4 working root directory, the default is.exe file directory, or linux execute prop file directory.

Function: Password (string)

Description: Set the network access password for C4, the default is DTC40@ZSERVER

UI function: Title (string)

Description: Can only work with C4's annotation UI template, set UI window title

UI function: AppTitle (string)

Description: Can only work with C4 annotated UI template, set APP title

UI function: DisableUI (string)

Description: Can only work with C4's annotation UI template, mask UI operation

UI function: Timer (unit millisecond)

Can only work with C4 annotation UI template, set the UI environment under the main loop millisecond cycle

C4 Help command

The Help command is the server maintenance + development debugging command built into C4. Whether it's console or ui, the help command is built in, and these commands are generic.

Command :help

The list of available commands is displayed

Command :exit

Alias :**close**
Description: Close the server

Command :service(ip address, port)

Reload parameter**: service(ip address)**
Reload parameter: **service()**
Aliases :**server**,**serv**
Server internal information report, including physical server information,p2pVM server, number of connections, traffic, server built-in startup parameters. If empty parameters will be simple report.

Command :tunnel(ip address, port)

Reload parameters: **tunnel(ip address)**
Overload parameter: **tunnel()**
Aliases :**client**,**cli**
Description: Client internal information report. If empty parameters will be simple report.

Command :reginfo()

Output has been registered c4 service,c4 each service will have the corresponding CS module, such as dp server +dp client.

Command :KillNet(ip address, port)

Reload parameters: **KillNet (ip address)**
Kill the corresponding c4 network service directly in the IDCfault way

Command :Quiet(Boolean)

Reload argument: **SetQuiet(**Boolean**)**
Switch the quiet mode, in the quiet mode, the server will not output the daily command execution status, but the error is prompted, such as the command model execution exception

Command :Save_All_C4Service_Config()

Instructions:

Immediately save the current server parameters, this is a server extension parameters, when c4 heap big after the server parameters too much, the maximum limit of the shell command line is 8192, under normal circumstances, simply can not write too many startup parameters in the shell command, so c4 provides a file form of parameter loading method, in default, and no parameter file

Through **Save_All_C4Service_Config()** can generate the suffix.conf server parameter file,.conf file is stored in the depnd subdirectory corresponding to the current server directory,.conf is a configuration file in ini format.

If you start c4 with.conf as the server argument, the command line arguments will be overwritten.

Save_All_C4Service_Config() is mostly used to deploy startup parameters when running the server for the first time, mainly to reduce the size of command line input. In real system integration, the command line reaches a 200,300 characters, which is very not easy to read and modify. Therefore, the.conf startup parameter is an important part of deploying c4.

Command: Save_All_C4Client_Config()

Note: Save all C4 guest parameters that are currently constructed immediately. The function is basically the same as **Save_All_C4Service_Config()**. In the system integration work, the client parameters are very few, which can be directly written into the shell command line, but if you want to beautify the command line, make it easy to read, then use the file parameters.

Command: HPC_Thread_Info()

Immediately output all **TCompute** thread instances in the current process,Z threads are created and executed using **TCompute**, and each thread will have a **thread_info** string identifier, used to identify the role of this thread. In the ZNet thread will be very many, there are **HPC** load thread,**CompleteBuffer background decoding coding thread**,**ZDB2 thread**. If you directly make RT library built-in **TThread**, then **HPC_Thread_Info()** will not output the thread state.

This command is mainly used for server debugging, analyzing performance bottlenecks, and finding bugs

Command :ZNet_Instance_Info()

Alias: **ZNet_Info()**
Description: Output all IO instances of ZNet immediately, including physical connections,p2pVM connections. It is used to diagnose the connection status and analyze the problems encountered when C4 enters the network.

Command: Service_CMD_Info()

Alias: **Server_CMD_Info()**
Immediately output the cpu consumption statistics of all command models in the server, these commands will be very many, hundreds. More used in the analysis of performance bottleneck positioning. **Service_CMD_Info()** also contains the number of times the command was sent, but not the cpu consumption of the command.

Command: Client_CMD_Info()

Alias: **Cli_CMD_Info()**
Immediately output all client command model cpu consumption statistics, and **Service_CMD_Info()** format is almost the same, because C4 is an interactive network, client statistics will reflect the server delay.

Command: Service_Statistics_Info()

Alias: **Server_Statistics_Info()**
Immediately output all the internal statistics of the server, including IO trigger frequency, encryption calculation frequency, main cycle frequency, the amount of data sent and received and other key information, the server will include physical server +p2pVM server

Command: Client_Statistics_Info()

Alias: **Cli_Statistics_Info()**
Note: Output all client internal statistics immediately, the output format is almost the same as **Service_Statistics_Info()**

Command: ZDB2_Info()

Immediately output ZDB2 database status,ZDB2 is a set of hierarchical architecture database system, at present ZDB2 has progressed to the third generation system, here **ZDB2_Info()** is also the output of the third generation of ZDB2 system, in C4 framework integrated FS2,FS, such services, all C4 services made in 2021, are the first The second generation of ZDB2 system, can not be **ZDB2_Info()** unified output state. The amount of information and design that **ZDB2_Info()** outputs from the third generation of ZDB2_Info() is very large and can only be covered in a single pass: it is a good way to use **zdb2_info ()** to see the status in the database and background monitored by the sixth generation.

Command: ZDB2_Flush()

The ZDB2 write cache immediately brush into the physical device, the use of information and **ZDB2_Info()** have a very large amount of information, here can only be taken: in the sixth generation of monitoring data background debugging array system hardware with the command, need to be combined with disk cache monitoring, memory monitoring, physical IO monitoring together. Function is to analyze the IO bottleneck of the array system.

ZNet kernel technology - lock multiplexing

Critical-Section is a hardware-based thread-locking technology for operating systems

The more threads a process contains, the more Critical Section will correspond to. In the system monitor, you can see the number of threads + the number of process handles. When the number of these two is large, the whole system may be unstable, at least in the process analysis or system crash, the thread + handle of the target process is two very important indicators.

Generally speaking, each thread will correspond to at least one or more Critical-Section handles, depending on the specific process writing.

The Z system kernel uses a multiplexing route for Critical-Section, and the ZNet server will see handle peaks in the monitor when running, but this is not a real Critical-Section, and you need to enter hpc_thread_info through the command c4 console to see the real Critical-Se ction and thread state.

ZNet kernel technology -Soft Synchronize

Soft Synchronize technology emulates the main line of rtl.

The design mechanism of ZNet relies heavily on the main line and therefore uses the Thread Synchronize system extensively. **In** the **asynchronous communication library** of **ZNet**, the **DIOCP/CrossSocket inter-thread Synchronize mechanism also uses** the **Thread Synchronize mechanism** to **control operations such as thread start and stop. Among them more or WaitFor thread mutual exclusion wait**, **if the queue is not completed**, **send the exit command to the thread** is easy to get stuck in it, **this time the problem is** often **by** the **outside program does not correctly clear the inter-thread execution call**, **clean up the inter-thread execution program this operation**, in **fact**, **CheckSynchronize**, this is a master Thread specific synchronization queue execution call. If the synchronization operation occurs in the thread, only the CheckSynchronize response is executed. In the vcl form system, this is automatically invoked by the application. To bypass the application, you need to master the main loop technique.

For example, if Thread.OnTerminate is given and CheckSynchronize is not given, no event is triggered.

Soft Synchronize solves the event transfer mechanism between threads and replaces CheckSynchronize.

Kernel :Check_Soft_Thread_Synchronize

The emulation main thread Synchronize code is executed, the RTL system Synchronize code is not executed. Z is all use the approach to perform the Synchronize code, including DIOCP/Cross/ICS8 / ICS9 / Indy/Synapse. **For example, when the asynchronous cross/diocp** library **uses waitfor**, **this causes** the Synchronize **of the main emulation line to run** during the **wait** process.

The RTL system Synchronize mechanism is used when the compilation switch Core_Thread_Soft_Synchronize is turned off.

The API supports programs that need to run in a dual-mainline environment.

Kernel :Check_System_Thread_Synchronize

The RTL system Synchronize code is executed, while the emulation main line Synchronize code is also executed

Automatically handles state, ignoring the compilation switch Core_Thread_Soft_Synchronize on or off.

This API supports programs to run in the main thread environment.

ZNet kernel Technology - Dual mainline

ZNet can open two mainlines simultaneously in a single process. When the dual-mainline model is started, the following things will happen:

- The entire ZNet system and the various libraries contained in ZNet all work in the submain thread.
- All RTL systems, including native lcl,vcl,fmx, all work in the original main thread.
- The submain thread and the primary thread will each maintain their own main loop, the main loop technology is omitted here
- Synchronize technology is used for the secondary main line and primary main line to synchronize data to each other
- The dual-mainline technology supports Linux programs built on win/android/ios and fpc
- With UI program, run the server will not have a feeling of stuttering
- 2. Running ZNet in a dll/ocx is equivalent to opening two EXEs, in which the exe and dll each go a main thread
- The submain thread can run http,c4,znet

RTL original main thread syncs to the submain thread

After the dual main thread mode, the data of the submain thread is accessed from the RTL main thread

```
TCompute.Sync(procedure
begin
  // Here access the data inside ZNet, including processing c4,cross,diocp,ics these communication data
end);
```

The submain thread goes to the original RTL main thread in the same step

After the dual main thread mode, the data of the RTL main thread is accessed from ZNet, which is to access VCL/FMX from ZNet

```
TThread.Synchronize(TThread.CurrentThread, procedure
begin
  // Here access and modify vcl/fmx,UI in these data
end);
```

Double mainline opens later main loop

The ZNet subthread API,**Check_Soft_Thread_Synchronize, is** located in the z.coor.pas library
RTL's original main thread API,**CheckSynchronize**, is located in the vcl-System.Classes.pas/lcl-classes.pas library

ZNet Performance Toolkit User Guide

CPS Toolbox =Caller Per second tool. All cps count cycles are 1 second. Located in the z.co.pas library.

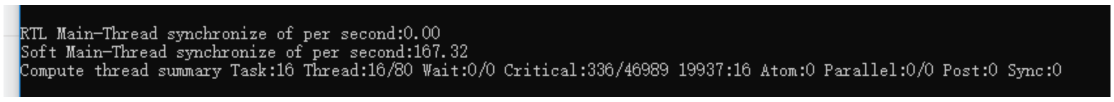
- **CPS_Check_Soft_Thread**: secondary main loop performance counter.
- **CPS_Check_System_Thread**:RTL main loop performance counter.

Access method, CPS_Check_Soft_Thread.CPS, which is the number of calls per second

All instances of ZNet have CPS performance counters built in to calculate how often the server main loop is called per second as well as cpu usage...

Performance bottleneck analysis

Start any C4 program, the command line type **hpc_thread_info**, get the following feedback



```
RTL Main-Thread synchronize of per second:0.00
Soft Main-Thread synchronize of per second:167.32
Compute thread summary Task:16 Thread:16/80 Wait:0/0 Critical:336/46989 19937:16 Atom:0 Parallel:0/0 Post:0 Sync:0
```

RTL Main-Thread synchronize of per second:0.00, number of times the RTL main loop is invoked per second

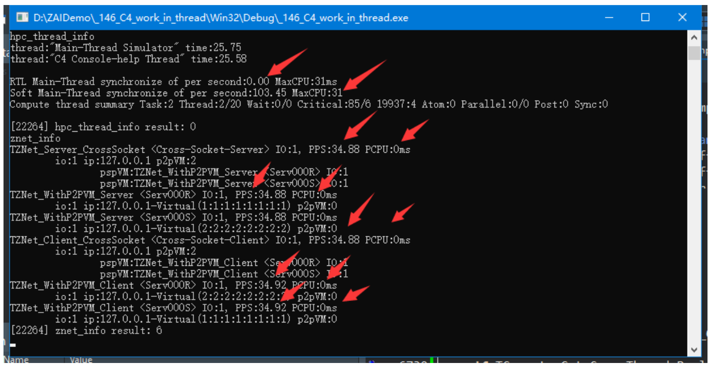
Corresponds to **CPS_Check_System_Thread**

Soft main-thread synchronize of per second:167.32, number of calls per second for the secondary Main loop

Corresponds to **CPS_Check_Soft_Thread**

Result: **This C4 program, using the secondary main loop technology, the main loop occurs 167 times per second, the higher the call frequency, the better the fluency. If the program occurs stutter, it is necessary to check the point of stutter: maybe some function occupies the conduction to the main loop, resulting in too low CPS, especially the program that opens the timer event.**

Click **znet_info** on the command line to get the following feedback



```
D:\ZAIDemo\_146_C4_work_in_thread\Win32\Debug\_146_C4_work_in_thread.exe
hpc_thread_info
thread:"Main-Thread Simulator" time:25.75
thread:"C4 Console-help Thread" time:25.58

RTL Main-Thread synchronize of per second:0.00 MaxCPU:31ms
Soft Main-Thread synchronize of per second:103.45 MaxCPU:31
Compute thread summary Task:2 Thread:2/20 Wait:0/0 Critical:85/6 19937:4 Atom:0 Parallel:0/0 Post:0 Sync:0

[22264] hpc_thread_info result: 0
znet_info
TZNet_Server_CrossSocket <Cross-Socket-Server> IO:1, PPS:34.88 PCPU:0ms
        io:1 ip:127.0.0.1 p2pVM:2
                pspVM:TZNet_WithP2PVM_Server <Serv000R> IO:1
                pspVM:TZNet_WithP2PVM_Server <Serv000S> IO:1
TZNet_WithP2PVM_Server <Serv000R> IO:1, PPS:34.88 PCPU:0ms
        io:1 ip:127.0.0.1-Virtual(1:1:1:1:1:1:1:1) p2pVM:0
TZNet_WithP2PVM_Server <Serv000S> IO:1, PPS:34.88 PCPU:0ms
        io:1 ip:127.0.0.1-Virtual(2:2:2:2:2:2:2:2) p2pVM:0
TZNet_Client_CrossSocket <Cross-Socket-Client> IO:1, PPS:34.88 PCPU:0ms
        io:1 ip:127.0.0.1 p2pVM:2
                pspVM:TZNet_WithP2PVM_Client <Serv000R> IO:1
                pspVM:TZNet_WithP2PVM_Client <Serv000S> IO:1
TZNet_WithP2PVM_Client <Serv000R> IO:1, PPS:34.92 PCPU:0ms
        io:1 ip:127.0.0.1-Virtual(2:2:2:2:2:2:2:2) p2pVM:0
TZNet_WithP2PVM_Client <Serv000S> IO:1, PPS:34.92 PCPU:0ms
        io:1 ip:127.0.0.1-Virtual(1:1:1:1:1:1:1:1) p2pVM:0
[22264] znet_info result: 6
```

PPS: Frequency of progress calls per second in ZNet

PCPU: Maximum cpu consumption of progress in ZNet
Result: Locate the instance in which the long process of the main loop is stuck. These instances can be servers or clients. After locating, use **Service_CMD_Info** and **Client_CMD_Info** to find the bottleneck of executing the command.

If the program does not use C4, you can use the API, **ZNet_Instance_Pool.Print_Status**, to output the status directly

Exclude ZNet overlapping Progress

First progress has an automatic anti-dead loop mechanism, the progress package progress will not be dead loop.

C4 has been optimized for progress overlap, and calling C40Progress each time ensures that Progress is only triggered once per ZNet instance

In non-C4 frameworks, Progress can be automatically triggered by p2pVM,DoubleTunnel, a main thread loop may cause 2-5 times progress, which is a pointless cost, if the server load is too heavy, repeated overlapping progress makes the shard load impossible to estimate accurately. This is an issue that must be addressed when it comes to precise optimization.

Use the following program paradigm

- Server.Progress;
- Server.Disable_Progress; **After the command is disabled, server.progress is not triggered later**
- other.progress;
- Server.Enabled_Progress;

The physical tunnel instance of p2pVM automatically traverses all p2pVM virtual connections within each progress.

When this is done, use **ZNet_Instance_Pool.Print_Status** to see the cps change. If the ZNet instance has a cps value similar to the main loop cps, then the progress is basically fine. Next, we test the connections, process the commands, and when all pass, we're done solving the overlapping progress problem.

Revere server main loop progress

Almost all server tuning efforts face the main loop problem, and there are a lot of solutions involved here

- **Threads**: If the main loop code supports thread-safety, then it is good to use threads, thread safety is not locked safe, but the thread + main loop will not be locked because the main loop is opened in the thread.
- **Sharding**: The sharding technique is to cut out the amount of computation and only run part of the main loop at a time
- **State machine**: The state machine is to let the main loop in a certain environment, directly omit and do not process some code. This is especially true when it comes to processes like for and while
- **Structure and algorithm optimization**: A large number of structures are processed in the main loop. Optimization of structures, such as hash and biglist, can effectively improve the efficiency of the main loop.

- **CPU instruction level optimization can be ignored**: such as sse,avx, this optimization is difficult to say, it is difficult to improve several times, can not be compared with algorithmic level optimization, algorithm optimization is generally started 10 times.

The main loop is the lifeblood of the main thread,90% of the server scheduler is used to complete the main thread, child threads and coroutines are mostly used to share some special tasks. For example, in the pas circle, many servers like a ui, and they can see the running status of the service at any time. The ui of this vcl/fmx/lcl route is running under the main thread, which will be deeply affected by the main loop.

And the fully threaded server, such as erlang,go, will have a thread scheduling problem, here will use many complex program mechanisms to control the problem of collaboration between threads, and the fully threaded computing server will not make the project become very smooth, let alone invincible, because the server bottom is affected by the hardware limit.

Main loop + child thread, the future will still be the main model of the server. No problem with the main loop can be equivalent to solving 50% of the server performance bottleneck!!

ZNet kernel technology: A brief introduction to the TCompute Threading model

TCompute is not a threading technology, but a canonical model for programming using threads.

TCompute has a very large downstream dependency system, and without TCompute, these downstream systems would fail

- Z-AI: The GPU driver in the AI system is thread-bound, and DNN-Thread technology is to open a thread in the GPU and CPU, and let it bind, **for example, when the identification will pass a graph to the gpu**, **which is** to **call** the **thread api, then execute cuda copy in** the **thread**, **and finally caller** The **gpu uses dnn** library to do parallel computation and get **results**. The **whole IO process is a threaded work**, and the process of recognizing IO by the main thread is mostly used when the demo demo api and mechanism are presented. The normal gpu program threading technology is used on a scale.
- AI toolchain: Since AI involves the field of big data, all data operations are unlikely to be completed instantaneously, so they are open threads, and its process is to **lock UI**, **run thread processing**, and **UI unlock.** The **most typical application is AI_Model_Builder**, **which often takes several minutes to complete a granular data operation**.
- ZDB1: This is an old chain database system, in 2017, a large-scale upgrade: the query flow of ZDB1 is packaged into pipe, and then the main thread synchronization mechanism is used for query scheduling. **As** a reminder**, when using zdb1, the query speed is at least doubled if the main thread Check_Soft_Thread_Synchronize (100) is used. The submain soft synchronize mechanism is more efficient than rtl library CheckSynchronize. However, the problems of ZDB1 are also obvious: in 2017, the depth of the thread was not mastered enough, resulting in the formation of a systematic solution, only a flash in the wind, mostly as a database companion, file packaging, installation program and other small functions to use**. **Can't go deep!**
- ZDB2: **can depth** the **technical system**, the **future is three-dimensional form**, the whole scheme as a data application model before and after 3 years in the design and improvement, just the foundation, has experienced 3 generations of application system, in the current advanced generic structure + advanced threading technology support, the development path can be basically clear: big data foundation support technology system, **in** the **ZDB2 system**, **a database**, **It can be run by 10 array disks, or 10 array systems under common load, and the technical scheme used to drive these large arrays is the thread, and each database file api works in an independent thread. ZDB2 in operation**, the **internal can range from tens to hundreds of threads**, even in the size of 10TB small database I/O threads can also eat >100GB memory +>20 core cpu, we do not use the array system thinking to look at ZDB2, data engine system and file can not be directly compared.

- Parallel programs: **Parallel programs are a necessary technical solution for modern processes, but parallel programs also have very obvious problems, the parallel support library used by fpc/d is not ideal: can not support the specified correlation core and hyperthreads, or even can not specify the number of parallel threads per start, and the biggest problem is the compatibility and parallel granularity model of the library, such as for and Line granularity can be divided parallel can also be folded parallel, these places must be unified in order to heap large, otherwise parallel program can only be used as a function point to solve the local acceleration problem, can not be used as a part of modern technology, because it can not heap large**. Kernel parallelism is superior to D system +LCL system in space + time complexity + mechanism.
- **The** sixth generation monitoring system: **the** sixth generation monitoring AI server side**,** a quasi gpu server + an AI server application**,** can **bring 80 channels of 4k video, and the calculation formula of the amount of data per second is :3840*2160*4*25*80= about 60G of data needed to be processed per second. This kind of abnormal data scale requires a very in-depth grasp and practice of the entire architecture of topology + switch + network + thread +NUMA+CPU+GPU in order to make a process plan. And this process can not be done in one step, you need to first from the local individual links, simulation verification, bottleneck and optimization, and then to combine the process**. The result of doing this is that the cost of computing power will be directly 5-10 times, which is from hell to heaven.
- **The** whole server system: **the server main cycle and thread is inextricably related, the main cycle once encountered with the amount of computation of the process card 30 seconds will be routine, people will always keep the server running, observe the state, followed by the optimization work, it can be said that the native TThread lack of stacking and scheduling mechanism, only the specification can meet the intermodulation between threads Mutual,TCompute is a thread model mined from the computer to the mechanism, which can really stack the thread system and unify the d/fpc specification**.

The kernel technology of ZNet: a brief introduction to the structure system

The structure here is not the data structure in computer science, but the basic structure system of algorithm application.

Algorithm application is a kind of computational engineering, which will need to be unified, standardized, explorable, regular, social, structural system is the source of computational engineering data, is the pre-design of the design algorithm. Because the algorithm program will require human to pay the price of time, once the time reaches a certain degree, the algorithm scheme may not be ideal. The architecture is like a preparation to solve. Without this preparation, the algorithm will become difficult, and the realization of any idea will take time, even a lot of time.

Take ZDB2 as an example, the core work of ZDB2 is only responsible for data IO, these IO are in a very complex scheduling work, the purpose of these work, is to provide the algorithm with the original data, the algorithm then converts the data into the data source, and then, is the calculation process. For example, loading 1 billion pieces of data from ZDB2, using the algorithm to do a key to 1 billion to do an acceleration, the process to achieve this thing is to use the universal structure, such as TCritical_Big_Hash_Pair_Pool, because ZDB2 IO is threaded, the need to lock the hash universal structure, at this time, in the IO inside to hash structure heap data pointer can complete a billion of the simplest millisecond level query. After going deeper, it is to extend, delete, modify, statistics and so on in the universal structure, and all of this is to use the process to operate the structure. When this thing is solved, it will complete the dedicated data engine, which is different from the general DB engine, the special engine is omnipotent, as long as you are willing, you can take the GPU to run sort+sum, and the calculation speed, performance, I think the general DB engine can only look at its back.

Take ZNet as an example, in the ZNet framework, there are a large number of queue mechanisms, such as progress inside the traversal TPeerIO is to first copy all the IO Pointers to a container, and then traverse the container, if the cpu time consumption is too large, it will exit, forming a fragment processing mechanism, to be processed next time progress will continue This IO container copies the container pointer again when all the container processing is completed, and starts the next shard. On the other hand,ZNet's send, receive, and other mechanisms also use containers, rather than a brainless copy into a stream. When the queue container this structure is used a lot, if the TList mechanism to deal with the queue is very useless, every time the extraction of the first queue, have to experience some copy, I believe that many people in the pas circle have complained about TList, but TList coherent 1D data space, once the queue extraction and deletion, optimize it can only change the pointer, otherwise it is copy This efficiency is very painful, and finally, the length limit of TList is basically unsolvable. The biggest advantage of TList is that it is simple to use, in small data scale projects such as UI, no problem, put it on the server or forget it... .. The data containers used by ZNet are chain structure, this chain structure is connected by a small block of memory through the next pointer, very suitable for queue extraction requirements, in the kernel library mostly TBigList, TOrderStruct to name, imagine, sticky packet process,1 packet 1k,10M data is 10000 queue packets, in high traffic,TList computing power and TBigList can not be compared, this gap, run out with the test program will be 2000 times.

Take the **Learn statistics method as an example**, the structure system can be regarded as the IO language, that is, the input of one structure, the output of another structure. For example, the classification algorithm, the principle of these algorithms can basically calculate the homework of junior high school students, first use some calculation methods, or random methods, to generate some seed form of data, and then do search and matching around seed, all classification algorithms, basically the same idea, change to change, perhaps this will be unimagination, but the process is often very simple! As for data access and output, this is the most troublesome calculation. In 600585, the complete classification process of the project is 45% pre-processing, 10% calculation and 45% post-processing. In summary, the proportion of structural system in the statistical process will be greater than that of the algorithm. We usually see to call an api, get solve, process direct package, package dozens of structure + class is also normal, this is no longer a statistical algorithm, is the solution, the internal is 1,2,3,4,5 sequence steps to solve the problem. Face recognition, recommendation algorithm is this model.

Take thread as an example, thread is a large system, modern programs do not use threads, and the communication between threads and threads, the structure written out of thin air is unable to stack up to do the project, thread intermodulation, etc.,TThreadPost,TSoft_Synchronize_Tool, these universal structures can basically be used 1-3 lines to solve each other Intermodulation, minimal use, otherwise how can the heap big?

The most frequently used large structures in the ZNet architecture: **TBigList<>**, **TBig_Hash_Pair_Pool<>**

Common small structures in ZNet architecture: **TOrderStruct<>**, **TAtom<>**, **TPair<>**

The generic structure of ZNet is commonly used in the Hash library **Z.HashList.Templet**

ZNet classic linked list library **z.Liistengine**

ZNet's kernel technology: a brief introduction to structural combinatorial punch

Taking SVM and K-Cluster as an example,SVM can calculate nonlinear algorithm rather representative, while KC prefers linear. The calculation process of both can be basically the same: input first, then do input preprocessing, obtain operator data, and then calculate the final classification result. **The dimensional part is the core idea of the whole SVM, focusing on traversing the single-dimensional data equation, through traversing to get the optimal plane can be divided into single-dimensional points, and then several single-dimensional combination is the hyperplane segmentation points, and then in the hyperplane from the**

largest to the smallest span cut, and then use the dimension to calculate clustering, the process of thinking is to take the encoding and decoding route This step can have a bunch of optimization measures, if it is not optimized svm will be a very simple process ideas, and svm can write thousands of lines, can also be dozens of lines to solve, because there is a dimension span, these spans can act as a kind of memory conditions, that is, training modeling, using the model this process,svm **can also do very simple,pas system** can **also be used dozens of lines To make svm automatic classification**, **the premise is that there must be** structural **support, before I refer to the classical** approach mainly from shogun(c++ classical svm origin project), such as interested in finding their own research. K-Mean derivation **is a random number to generate centroid classification, cluster all** nearby, and **then define a new centroid** to **re-go through the process, iterating several times** to **complete** the **clustering.** At the structural level, as long as the definition, input and output, the rest can be directly handed over to open source various computing libraries to do. We usually find information on the Internet, a variety of complex formulas, which is a kind of expression language on the idea, the algorithm process, especially the main part of the idea is not too complicated. It is difficult to understand the non-line field, this field if the self-created things in the calculation, there can be hundreds of algorithms, and the core idea of the non-line is to do decoding and coding preprocessing of the data.

Take high-speed range search as an example, this field has not yet had time to write demo, its goal is to solve the scope of rapid search, for example, the amount of data to 1 billion, and the data at any time in the increase and deletion, to search the time range and coordinate range, if not to the algorithm violence traversal, perhaps a process will go for several minutes. The more effective way is to cut the time and coordinate range according to the metric, such as the file coordinates of the disk array, you can cut out a region for hash every 1M, and the processing range takes 1M as a small span to remember, the same way, time segmentation, you can cut according to the cutting, you can also cut on time, after the cutting only need to enter once you can accurately locate. ZNet's approach is to cache pointer,hash span, reference library for Z.ashminutes. Templet, implementation and combination of the main use of TBig_Hash_Pair_Pool<>+TBigList<>. Which time range acceleration algorithm, mainly used to search monitoring fragments, basically all can be searched in seconds, coordinate range acceleration algorithm, mainly solve simulation write cache, which is to simulate write files and ensure read and write consistency of the function, need to read and write in high-speed environment, such as write 100 length coordinates in 1024 position, this time need to find a series of cache 1024 position part buffer, so as to complete the file read and write data consistency.

Take bidirectional matching algorithm pairs as an example: bidirectional, when a pairing is completed, it traverses all goals. If optimization is not considered, pairing 1000 times than 1000, the computational amount is tens of millions. When pairing is completed, the paired data is eliminated, the computational amount will be much smaller, and with threading and parallel means, two-way pairing can be very fast. To solve the problem of deleting pairing,TList can not be solved,TList will reconstruct array buffer, which is very cpu consuming, you must use TBigList<>.

Take parallel sorting as an example: I don't know the fastest parallel algorithm on earth, my method is to first separate the storage and then arrange, for example, 1-10, 11-20 into separate blocks, and then use parallel rows of granular blocks, and finally row the whole block + to build the output. Structure is the use of TBigList<>, only TBigList can support a large number of such as 1 billion, directly using the memory pointer will make the sort very complicated.

Review: Designing the Universal structure TBigList<>

TBigList was not designed overnight. This comes down to a bit of history

1. In 2015, the original predecessor of TBigList was THashList, located in the Z.Liengine library, which was the first Hash library written by the author, and used TList extensively for transformation saving function
2. Around 2016-2017,THashList appeared the requirement of sequence restoration, and it was impossible to recall what sequence was restored. In short, after add 123, it should be possible to restore the order of 123 directly from THashList. This demand led to the transformation of the internal structure of THashList from simple to chain.
3. In 2020, TOrderStruct appeared. At that time, it was decided to gradually transplant the whole system code from classical structure to universal structure.
4. In 2020,TBigList was written. When the TBigList program came out, the test case was written very well, and along with it, the stubborn 100-year bug of THashList was also fixed, which was a very major fix. After that, TPair<>, TBig_Hash_Pair_Pool<>, were written successively
5. In 2021, TList will be deleted from all ZNet code and replaced with TBigList<> universal structure.
6. One day in 2021, I was bored, and I wrote a small demo of BigList pk TList. TList used all the optimal methods of delete, append and modify to perform a one-time energy pk with BigList. As a result, the processing power of BigList was almost 2000 times that of TList

TBigList design ideas: **first of all to solve the high-speed queue +Int64 level data amount + can be in large-scale cyclic code kind of stacking programming: must solve for this cycle needs. Among them, to solve the for cycle needs even higher than the requirements of performance, simply put, for must be a very simple process model, can not use anonymous functions**. Finally,TBigList was designed into today's universal structure, on this basis, later,ZDB2, a variety of new algorithms, new structural system, emerged, these structures and algorithms are too many,Z-AI system directly do not need to mention.
ZDB2 was written, in fact, it has announced that autonomous technology has entered the era of big data. The next step is to iterate over time.

Review: Designing the scripting engine ZExpression

Very, very long ago, the compiler has been the author's heart regret, the original idea: theory understand a pile, if you can not write once, the theory will be empty talk, this matter must have personal experience, eating, drinking and playing **is meaningless (today's feeling is no money is meaningless**).

- String parsing is the first problem facing, string parsing program is very complex, the beginning is to do word segmentation + word segmentation function, and then, began to try to do symbol parsing, inverse Poland, add, subtract, multiply and divide, the various characters into a prototype structure. Finally, the solution to the lexical conversion.
- The second step is to start trying to solve lexical semantic structures, such as lexical legitimacy, where the lexical structure is a tree. Because the lexical structure of 1+(1-2) is equivalent to 1+a, and the a structure is (1-2).
- The third step, began to try to translate the lexical structure into the executable program opcode code, this mechanism, using the simulation opcode code to achieve, the mechanism and x86 decoding work in the same way, the difference is different code table.
- The fourth step begins to greatly strengthen the Parsing support stages, with an infinite approach to bison+yacc. The technical details are omitted here and will be discussed in the next section.
- The fifth step is to start using the ZExpression system :C4 startup,ZAI script, Generation 6 script,pascal code rewrite model, all of which are included in the ZExpession system.

Mother transplant technique for ZNet: Z.Persing

ZNet is a giant code project as the parent + carrier, these codes are actually generated by machine compilation technology analysis + reconstruction, we usually use prp transplant ZS, or prp upgrade ZNet, which is a data model + analytic reconstruction technology at work.

The analysis + reconstruction technology in the compiler, its idea draws on the bison+flex/lex+yacc system.

Take a simple particle

if(1+1=2) kill;

if 1+1 = 2 do kill

These are two very different lexical bodies, and in the bison/yacc system we use code expressions to describe these lexical bodies, which is the scripting language, and then we use the coding process to get them to form unified structural data that can express the flow.

In the Z.Perl system, there is a mechanism that can be used to win the trick, the probe technique

The probe technology is based on the Parsing of parts of speech. The parsing system will first parse numbers, symbols, floating point, strings, remarks,ASCII, etc., and they will be divided into parts of speech to form a part of speech chain structure.

For example if the kill (1 + 1 = 2), its part of speech for chain: ASCII, symbol, num, symbol, num, symbol... .

Probe technology, is to make an approximate judgment of the part of speech chain, and then enter the branch process.

Probe technology can distinguish different parts of speech structure combinations + lexical bodies, it will form conditional paradigms, these conditional paradigms, is the design idea of bison/yacc.

When the lexical program has conditional detection, it can use the ant to climb while using the probe conditional paradigm to open branch program to deal with the randomness of the handwritten morphology.

It is also with the powerful probe mechanism,pascal rewriting model technology can correctly and completely parse the pas code and reconstruct the desired object code.

In another direction, internationalized remarks and string machine translation technology, is also the use of Z.Perl system, and string translation is very simple and violent, is to find parts of speech for the string and remarks data structure translation, and then re-constitute different languages. International project free you can go to https://github.com/PassByYou888/zTranslate

Z.perl's probes are flexible and can be hit wherever you want, and you can do things in the ant program that bison/yacc can't do. At this point**, Z.Perl is the technical system that rules cash flow: as long as the person who uses Z.Perl can write his own language correctly, and he can**

make the collective consistent use of his predictions, then he will become the godfather of software, games, and programs, which will give him more power than any investor, general manager, or chairman Absolute technical power. **If the company is valued at $10 million, I wouldn't be surprised if the guy who used Z.Pos to build the company's production system would be worth half of it**. Because the company depends on the product, the product depends on the collective production, and the core technology of production depends on the user of Z.Perl.

On the other hand, Z.Perl system, no matter HTML,JS,Pas,C++,XML, can do it.

How do I overturn a project using ZNet

If you used ZS to go physical dual channel project, overturn directly to C4.

If non-C4 dual channels were previously used, override direct change to C4.

If you previously used ics,indy,win socket for non-Web items, directly reverse to C4, full crush

If the project is already C4, you can change the registration name, **RegisterC40('MY_Serv', TMY_Serv, TMY_Cli)**, and then start a new project based on C4 **RegisterC40('MY_Serv2.0', TMY_Serv2, TMY_Cli2)**. Simply put, change the increment, add a version number to the unit name, and Reg the new service on the line.

2. How to use ZNet to develop web projects

Data communication layer directly run ZNet,UI layer with webapi access to ZNet projects can be. For example, Post+Get can basically cover 90% of webapi needs, and ZNet comes with a webapi demo project.

ZNet with http and web

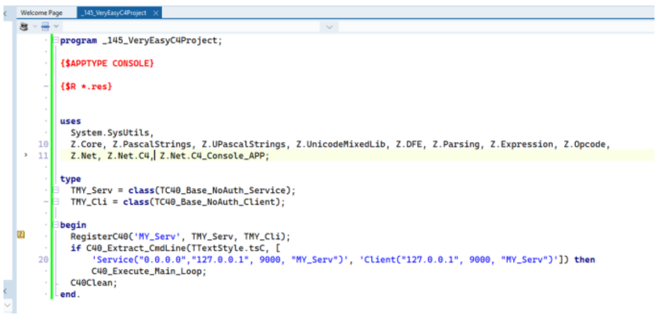
ZNet= Large CS server

http= Communication protocol

web= Global Wide Area Network, Internet

web contains ZNet, in the web environment with apache,nginx bridge communication module of large websites abound, perhaps readers can try to use the second-level domain name or domain server to do the bridge module and shitter, if the internal involves big data or complex protocol, directly use ZNet to do a communication layer package api to the web.

Text finally to a minimalist C4 CS demo



```
program _145_VeryEasyC4Project;

{$APPTYPE CONSOLE}

{$R *.res}


uses
  System.SysUtils,
  Z.Core, Z.PascalStrings, Z.UPascalStrings, Z.UnicodeMixedLib, Z.DFE, Z.Parsing, Z.Expression, Z.Opcode,
  Z.Net, Z.Net.C4, Z.Net.C4_Console_APP;

type
  TMY_Serv = class(TC40_Base_NoAuth_Service);
  TMY_Cli = class(TC40_Base_NoAuth_Client);

begin
  RegisterC40('MY_Serv', TMY_Serv, TMY_Cli);
  if C40_Extract_CmdLine(TTextStyle.tsC, [
      'Service("0.0.0.0","127.0.0.1", 9000, "MY_Serv")', 'Client("127.0.0.1", 9000, "MY_Serv")']) then
      C40_Execute_Main_Loop;
  C40Clean;
end.
```

Complete the text.

The 2023-10-6

by.qq600585