理解 C4 的高流量和高并发支持机制

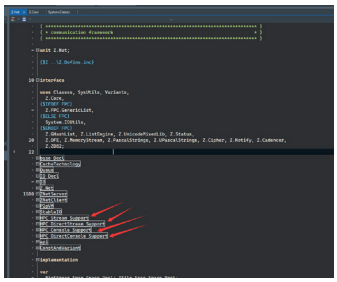
早期设计 VCL 时处于对 UI 的统一化支持机制，VCL 消息处理中枢，也就是 UI 调度，管理，使用的中枢，都集中于主线程，也就是 system 库里面 MainThreadID（主线程）。

这是因为 VCL 设计定位要走，傻瓜，简单，易用，堆大，这种路线，因此就形成了主线程+子线程这种程序机制。

由此，就引发了，TThread.Synchronize，这类同步机制，这是 VCL 独有的。这也是许多 VCL/FMX/LCL 程序都会共同面临的问题：单一化，死板化的主线程同步机制。

C4 依赖于 Z.Net.pas 库，该库的核心网络调度机制，所有的收发事件，都会集中于主线程触发，因此高流量+高并发，不管收发，都吃满主线程。**这时候，Z.Net 服务器对于接收请求的做法是用子线程分流处理，而发送数据的做法，则需要自己控制调度，需要自己想法子让数据全部通过主线程发出去**。

先说，服务器接收请求的分流支持技术，打开 Z.Net.pas 库，解决方法在以下折叠代码中，思路上是把请求数据复制给一个子线程，然后启动线程，待子线程处理结束，再把数据反馈给请求方，整个流程自动化进行。目前数据结构只能支持 TDataFrameEngine，因此面对 CompleteBuffer，BigStream 这类机制，需要自己开子线程来处理。



接下来是发送数据，这一部分，涉及面很多，不管怎么处理，我都建议先理解线程和数据结构支持机理。

Z.Core.pas 库提供了 TCompute 线程支持，它横跨 FPC/Delphi 的支持机制，并且 TCompute 使用 MT19937 统一化了随机数支持机制（算法并行化大难题）。关于 TCompute 机制有许多 Demo，在那些 demo 中也撰写了许多讲解备注，通过阅读，动手尝试，就能搞明白 TCompute。

Z.Core.pas 库也提供了 TOrderData，TBigList，这类可以支持高速队列机制的数据结构，并且都具有线程无关性机理，有许多相关 Demo。

Z.IOThread.pas 库提供了大数据的数据队列处理机制，并且它是并行化处理的，也有许多相关 Demo。

回到发送数据，当面对高流量+高速收发时，**思路上应该明确+清晰数据流从那里发出，处理过程怎么走，最后把数据汇聚到主线程通过 ZNet 发出去**。

一般来说，大数据块，用 ZNet 的 BigStream 机制，小数据块(低于 10M)，用 CompleteBuffer，主要问题在于：有没有把数据的生成，处理，流向，这些逻辑思考清晰。只要把 TCompute，TBigList，TIO_Thread，用法+使用场景搞明白，发送是不会构成问题的。

by.qq600585
2022-10-15