序列化的命令队列机制详解

目录

序列化的命令队列机制详解 1
ZS 中的 CommunicationFramework 内核包含如下命令模式 1
SendConsoleCmd 1
SendStreamCmd 1
SendDirectConsoleCmd 2
SendDirectStreamCmd 2
SendBigStream 2
SendCompleteBuffer 2
总结 2
客户端发出一批带有反馈的命令后，接下来发生了什么事？ 3
当 SyncOnResult=False 时（默认设置，匿名函数支持） 3
当 SyncOnResult=True 时（严格顺序流，不能使用匿名函数） 4

ZS 中的 CommunicationFramework 内核包含如下命令模式

SendConsoleCmd

文本模式的命令，服务器处理后会有反馈，适用于请求，性能低
命名解释：Send Console Cmd，发送一条纯文本内容的命令，文本会经过 TdataFrameEngine 的编码，用压缩和加密

SendStreamCmd

数据结构模式的命令，服务器处理后会有反馈，适用于请求，性能低
命名解释：早期的 ZS 框架，根本没有自动化处理 TDataFrameEngine，都是原始的 Stream 数据，每次收发时，要使用 TdataFrameEngine 对其编码解码，后来引入了 TdataFrameEngine 后，这种发送发送 Stream 就变成了发送数据结构，但是命名关系到若干工程的兼容性，所以仍然沿用了含有 Stream 字样的命名

SendDirectConsoleCmd

文本模式的命令，无反馈，适用于广播，性能优于请求

SendDirectStreamCmd

数据结构模式的命令，无反馈，适用于广播，性能优于请求
命名解释：早期的 ZS 框架，根本没有自动化处理 TDataFrameEngine，都是原始的 Stream 数据，每次收发时，要使用 TdataFrameEngine 对其编码解码，后来引入了 TdataFrameEngine 后，这种发送发送 Stream 就变成了发送数据结构，但是命名关系到若干工程的兼容性，所以仍然沿用了含有 Stream 字样的命名

SendBigStream

发送大型 Stream 的命令，无反馈，适用于大型文件收发

SendCompleteBuffer

发送大型内存块的命令，无反馈，适用于高速通讯

总结

所有命令在发送后，服务器在任何时候，都会依次按发送的顺序执行，在不修改参数的情况下，不会发生乱序列情况。影响服务器执行序列的只有 SendCompleteBuffer 命令，如果在服务器端需要它对匿名函数支持，需要关闭 SyncOnCompleteBuffer，默认情况它是打开的，我们把 SyncOnCompleteBuffer:=False(使用时手动关闭即可) 。一旦关闭，CompleteBuffer 的触发事件，就会变成后置式，如果服务器的模型设计指令序列有要求，关闭 SyncOnCompleteBuffer，将会发生错误

客户端发出一批带有反馈的命令后，接下来发生了什么事？

当 SyncOnResult=False 时（默认设置，匿名函数支持）

这时候，命令的反馈都是以后置式方式触发的，性能相比立即触发略低，这种方式主要是自动化的避免死锁情况。试想，假如在客户端，一条命令反馈的触发事件中，要实现发送另一条命令，这是一件方便的事情。在 Delphi 常用的匿名函数，主要在依靠后置事件。
举例

- SendConsoleCmd(命令 1，回调函数 1)　　//发送一条带有反馈的命令
- SendConsoleCmd(命令 2，回调函数 2)　　//发送一条带有反馈的命令
- SendConsoleCmd(命令 3，回调函数 3)　　//发送一条带有反馈的命令

在默认情况下，服务器会按次序运行命令（服务器无影响），但是客户端的反馈顺序是

- 命令 1 发送
- 命令 2 发送
- 命令 3 发送
- 执行回调函数 1
- 执行回调函数 2
- 执行回调函数 3

说明，客户端的反馈总是后置式，当一个命令批次被处理完成后，后置式的反馈事件才会开始工作。这种技术，我也将它称之为 Delphi 匿名函数支持。

当 SyncOnResult=True 时（严格顺序流，不能使用匿名函数）

这时候，命令的反馈都是以立即式方式触发的，这种方式经常会发生避免死锁情况，其问题是出在触发点都在线程中，假如一个线程正在执行 Sync(事件)，而在事件中，你又发出一条等待反馈的命令，那么就是 Sync(Sync(事件))，这就是死锁的根源。解决死锁的办法有 2，要么就用 PostProgress.PostExecute 抛出一个后置事件在回调中去执行另一个等待回调，要么就把 communicationFramework.SyncOnResult 关闭掉

举例

- SendConsoleCmd(命令 1，回调函数 1)　　//发送一条带有反馈的命令
- SendConsoleCmd(命令 2，回调函数 2)　　//发送一条带有反馈的命令
- SendConsoleCmd(命令 3，回调函数 3)　　//发送一条带有反馈的命令

在默认情况下，服务器会按次序运行命令（服务器无影响），但是客户端的反馈顺序是

- 命令 1 发送
- 执行回调函数 1
- 命令 2 发送
- 执行回调函数 2
- 命令 3 发送
- 执行回调函数 3

说明，客户端的命令不会一次全部发送完，而是发送 1，执行反馈，发送 2，执行反馈，发送 3，执行反馈

By qq600585
2018-10-23