基于动态库的通用应用集成框架

# 动态库并发调用

## 结论

多个程序引用同一个动态库，单例仅存在于各自的进程中，不同进程中的单例对象互不影响。

## 测试工程

见SingletonLib测试工程，启动两个TestConsole控制台（引用的是同一个单例类SingletonLib），通过输入“+”、“-”分别进行计数，两个控制台中SingletonLib对象的计算互不干扰。

## DLL调用机制

1. 微软支持

参考资料：微软支持网页

<https://support.microsoft.com/zh-cn/help/815065/what-is-a-dll>

内容：

“通过使用DLL，程序可以实现模块化，由相对独立的组件组成。例如，一个计帐程序可以按模块来销售。可以在运行时将各个模块加载到主程序中（如果安装了相应模块）。因为模块是彼此独立的，所以程序的加载速度更快，而且模块只在相应的功能被请求时才加载。”

1. Windows DLL基本原理与加载连接的实现

<https://blog.csdn.net/zhenyongyuan123/article/details/4063749>

“DLL是一种磁盘文件（通常带有DLL扩展名，是标准win32可执行文件－“PE”格式），它由全局数据、服务函数和资源组成，在运行时被系统加载到进程的虚拟空间中，成为调用进程的一部分，进程中所有线程都可以调用其中的函数。如果与其它DLL之间没有冲突，该文件通常映射到进程虚拟空间的同一地址上。……**在Win32环境中，每个进程都复制了自己的读/写全局变量。如果想要与其它进程共享内存，必须使用内存映射文件或者声明一个共享数据段**。DLL模块需要的堆栈内存都是从运行进程的堆栈中分配出来的。”

从原理上多进程调用DLL彼此互不影响。

# 主键必要性

## 主键必要性

参考资料：Vortex OpenSplice DDS IDL PreProcessor User Manual

内容：第7.1节 Defining Keys

“The identifier <key> is the member of a struct. For a struct either no key list is defined, in which case no specialized interfaces (TypeSupport, DataReader and DataWriter) are generated for the struct, or a key list with or without keys is defined, in which case the specialized interfaces are generated for the struct.”

首先keylist是需要在IDL中定义的，否则IDL编译器不会对主题结构体创建相应的TypeSupport、DataReader、DataWriter。但是keylist后面的key不一定需要定义，如下：

|  |
| --- |
| struct SYNodeDetect  {  string node\_name;  long msg\_type;  long node\_type;  };  #pragma keylist SYNodeDetect |

SYNodeDetect后可以不设置具体的key。这种情况下IDL编译器依然会对主题结构体创建相应的TypeSupport、DataReader、DataWriter，与此同时也可避免所有数据都要定义主键的问题。对于没有主键的数据，样本之间

目前我们公司的仿真引擎采用的就是上面的方式来定义keylist，详见SimData.idl文件。

# 数据持久性和正确性

## 主题

主题（topic）：主题是DDS程序信息交互的单元，主题由名称、类型和一系列Qos策略组成。

实例(instance)：对于不带key的主题，一个主题就是一个实例；对于带key的主题，一个key对应一个实例。

样本(sample)：主题对应的一条数据。

## 影响数据持久性和正确性的因素

### QoS

1. DURABILITY QoS

|  |  |
| --- | --- |
| **策略** | **含义** |
| VOLATILE | 一旦发布，dds不再保存数据；后加入的节点无法获取先前的数据 |
| TRANSIENT\_LOCAL | 发布者本地存储发布的数据；如果发布者依然存在，那么后来加入的订阅者可以获取先前发布的数据。 |
| TRANSIENT | 全局数据区保存数据以便后来加入的订阅者可以获取先前的数据。 |
| PERSISTENT | 数据被永久保存（比如保存到硬盘），即使系统重启数据仍然可用 |

1. RELIABILITY QoS

|  |  |
| --- | --- |
| **策略** | **含义** |
| BEST\_EFFORT | 丢失的数据不会重发 |
| RELIABLE | dds会尝试发送所有的样本数据，丢失的数据会重发 |

1. HISTORY QoS

|  |  |
| --- | --- |
| **策略** | **含义** |
| KEEP\_LAST | dds会保存最新的若干条数据直到数据被take取走 |
| KEEP\_ALL | dds会尝试保存所有数据直到数据被take取走 |

### 读取数据的方式

DataReader读取数据的方式有2种：read和take；read读取数据后原来的数据仍然存在；take读取数据以后数据就不存在了。

# QoS方案

HISTORY 策略选择KEEP\_ALL，保证所有数据都不会被覆盖，数据读取方式选择take，确保旧数据及时清空，释放空间。

针对不同的数据传输效率、可靠性要求，提出三种QoS策略组合方案，如下。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **方案名** | **DURABILITY** | **RELIABILITY** | **HISTORY** | **数据读取方式** | **备注** |
| 方案一 | VOLATILE | BEST\_EFFORT | KEEP\_ALL | take | 效率最高，可能丢失数据，后加入的节点无法获取先前的数据 |
| 方案二 | TRANSIENT\_LOCAL | RELIABLE | KEEP\_ALL | take | 平衡效率和可靠性，后加入的节点不一定能获取先前数据 |
| 方案三 | TRANSIENT | RELIABLE | KEEP\_ALL | take | 可靠性相对最高，后加入的节点可以获取先前数据 |

# 通用应用集成框架

应用启动时，需要将initTool、setToTool、setFinish和endTool四个回调句柄通过dllStart接口发送到发送Edge DLL中，回调函数具体定义间上左图，分别表示初始化应用程序、步长内发送一个数据到应用程序、步长内所有数据发送完成和结束与应用程序关联；动态库接口除了接收配置文件和回调句柄的dllStart外，还包括dllSetValue、dllAdvance，这三个接口分别表示初始化动态库、步长内发送一个数据到动态库（DDS域）、步长内所有数据发送完成并请求推进。

仿真运行中的应用、Edge DLL和虚拟DDS总线交换关系如上右图所示，该过程能够实现应用初始化、应用请求推进、DDS允许推进、步长开始时DDS向应用推送数据和步长结束时应用向DDS推送数据过程，实现多学科应用通过Edge DLL统一加入DDS域进行仿真。

