Marshaling won't get in your way if it isn't needed

devblogs.microsoft.com/oldnewthing/20140926-00

September 26, 2014

Raymond Chen

I left an exercise at the end of last week's article: "Why is the `RPC_X_NULL_REF_POINTER` error raised only sometimes?" COM subscribes to the principle that if no marshaling is needed, then an interface pointer points directly at the object with no COM code in between. If the current thread is running in a single-threaded apartment, and it creates a COM object with thread affinity (also known as an "apartment-model object"; yes, the name is confusing), then the thread gets a pointer directly to the object. When you call `p->QueryInterface()`, you are calling directly into the `QueryInterface` implementation provided by the object. This principle has its pluses and minuses. People concerned with high performance pretty much insist that COM stay out of the way and get involved only when necessary. They consider it a plus that if there is no marshaling involved, then all pointers are direct pointers, and calls go straight to the target object without a single instruction of COM-provided code getting in the way. One downside of this is that every object is responsible for its own compatibility hacks. If there are bugs in the implementation of `IUnknown::QueryInterface`, then each object is on its own for working around them. There is no opportunity for the system to enforce correct behavior because there is no system code running. Each object becomes responsible for its own enforcement. Therefore, the answer to "Why is the `RPC_X_NULL_REF_POINTER` error raised only sometimes?" is "The marshaler is involved only sometimes."

If the object being called belongs to the same apartment as the thread that is calling into it, then there is no marshaler, and the call goes directly to the object. Since there is no marshaler, the marshaler isn't around to enforce marshaling rules. It's up to the object to enforce marshaling rules, and if the object chooses not to, then you get into the cases where a method call works when the object is unmarshaled and fails when the object is marshaled.

Raymond Chen

Follow