Location Transparency

One of the useful features of the Ice run time is that it is *location transparent*: the client does not need to know where the implementation of an Ice object resides; an invocation on an object automatically is directed to the correct target, whether the object is implemented in the local address space, in another address space on the same machine, or in another address space on a remote machine. Location transparency is important because it allows us to change the location of an object implementation without breaking client programs and, by using IceGrid, addressing information such as host names and port numbers can be externalized so they do not appear in stringified proxies.

For invocations that cross address space boundaries (or more accurately, cross communicator boundaries), the Ice run time dispatches requests via the appropriate transport. However, for a proxy invocation in which the proxy and the servant that processes the invocation share the same communicator (so-called *collocated* invocations), the Ice run time, by default, does not send the invocation via the transport specified in the proxy. Instead, collocated invocations take a short-cut inside the Ice run time and are dispatched directly.

Note that if the proxy and the servant do not use the same communicator, the invocation is *not* collocated, even though caller and callee are in the same address space.

The reason for this is if collocated invocations were sent via TCP/IP, for example, invocations would still be sent via the operating system kernel (using the back plane instead of a network) and would incur the full cost of creating TCP/IP connections, marshaling requests into packets, trapping in and out of the kernel, and so on. By optimizing collocated requests, much of this overhead can be avoided, so collocated invocations are almost as fast as a local function call.

For efficiency reasons, collocated invocations are not completely location transparent, that is, a collocated call has semantics that differ in some ways from calls that cross address-space boundaries. Specifically, collocated invocations differ from ordinary invocations in the following respects:

- Collocated invocations are dispatched in the calling thread instead of being dispatched using the server's concurrency model.
- The object adapter holding state is ignored: collocated invocations proceed normally even if the target object's adapter is in the holding state.
- For collocated invocations, classes and exceptions are never sliced. Instead, the receiver always receives a class or exception as the derived type that was sent by the sender.
- If a collocated invocation throws an exception that is not in an operation's exception specification, the original exception is raised in the client instead of `UnknownUserException`. (This applies to the C++ mapping only.)
- Class factories are ignored for collocated invocations.
- Timeouts on invocations are ignored.
- If an operation implementation uses an in parameter that is passed by reference as a temporary variable, the change affects the value of the in parameter in the caller (instead of modifying a temporary copy of the parameter on the callee side only).
- Asynchronous semantics are not supported for collocated invocations.

In practice, these differences rarely matter. The most likely cause of surprises with collocated invocations is dispatch in the calling thread, that is, a collocated invocation behaves like a local, synchronous procedure call. This can cause problems if, for example, the calling thread acquires a lock that an operation implementation tries to acquire as well: unless you use recursive mutexes, this will cause deadlock.

The Ice run time uses the following semantics to determine whether a proxy is eligible for the collocated optimization:

- For an indirect proxy, collocation optimization is used if the proxy's adapter ID matches the adapter ID or replica group ID of an object adapter in the same communicator.
- For a well-known proxy, the Ice run time queries each object adapter to determine if the servant is local.
- For a direct proxy, the Ice run time performs an endpoint search using the proxy's endpoints.

When an endpoint search is required, the Ice run time compares each of the proxy's endpoints against the endpoints of the communicator's object adapters. Only the transport, address and port are considered; other attributes of an endpoint, such as timeout settings, are not considered during this search. If a match is found, the invocation is dispatched using collocation optimization. Normally this search is executed only once, during the proxy's first invocation, although the proxy's connection caching setting influences this behavior.

Collocation optimization is enabled by default, but you can disable it for all proxies by setting the property `Ice.Default.CollocationOptimized=0`. You can also disable the optimization for an individual proxy using the factory method `ice_collocationOptimized(false)`. Finally, for proxies created from a property using `propertyToProxy`, the property *`name`*`.CollocationOptimized` configures the default setting for the proxy.

See Also

- IceGrid
- Threads and Concurrency with C++
- Connection Establishment
- Proxy Methods
- Obtaining Proxies