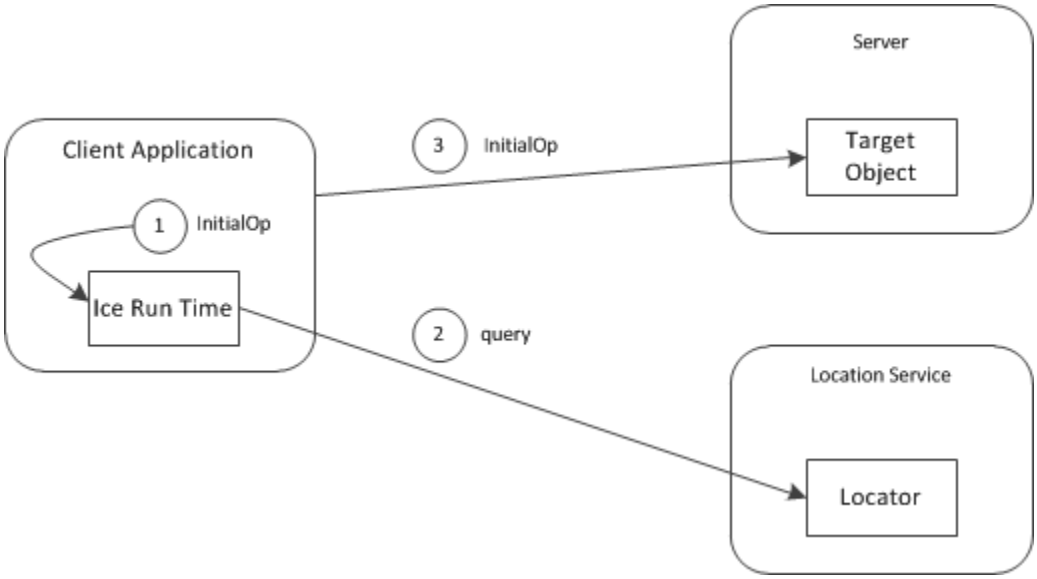
Locator Semantics for Clients

On this page:

Invocations with an Indirect Proxy

On the first use of an indirect proxy in an application, the Ice run time may issue a remote invocation on the locator object. This activity is transparent to the application, as shown below:



Locating an object.

1. The client invokes the operation `initialOp` on an indirect proxy.
2. The Ice run time checks an internal cache (called the *locator cache*) to determine whether a query has already been issued for the symbolic information in the proxy. If so, the cached endpoint is used and an invocation on the locator object is avoided. Otherwise, the Ice run time issues a locate request to the locator.
3. If the object is successfully located, the locator returns its current endpoints. The Ice run time in the client caches this information, establishes a connection to one of the endpoints, and proceeds to send the invocation as usual.
4. If the object's endpoints cannot be determined, the client receives an exception. `NotRegisteredException` is raised when an identity, object adapter identifier or replica group identifier is not known. A client may also receive `NoEndpointException` if the location service failed to determine the current endpoints.

As far as the Ice run time is concerned, the locator simply converts the information in an indirect proxy into usable endpoints. Whether the locator's implementation is more sophisticated than a simple lookup table is irrelevant to the Ice run time. However, the act of performing this conversion may have additional semantics that the application must be prepared to accept.

For example, when using IceGrid as your location service, the target server may be launched automatically if it is not currently running, and the locate request does not complete until that server is started and ready to receive requests. As a result, the initial request on an indirect proxy may incur additional overhead as all of this activity occurs.

Replication with a Locator

An indirect proxy may substitute a replica group identifier in place of the object adapter identifier. In fact, the Ice run time does not distinguish between these two cases and considers a replica group identifier as equivalent to an object adapter identifier for the purposes of resolving the proxy. The location service implementation must be able to distinguish between replica groups and object adapters using only this identifier.

The location service may return multiple endpoints in response to a locate request for an adapter or replica group identifier. These endpoints might all correspond to a single object adapter that is available at several addresses, or to multiple object adapters each listening at a single address, or some combination thereof. The Ice run time attaches no semantics to the collection of endpoints, but the application can make assumptions based on its knowledge of the location service's behavior.

When a location service returns more than one endpoint, the Ice run time behaves exactly as if the proxy had contained several endpoints. As always, the goal of the Ice run time is to establish a connection to one of the endpoints and deliver the client's request. By default, all requests made via the proxy that initiated the connection are sent to the same server until that connection is closed.

After the connection is closed, such as by Active Connection Management (ACM), subsequent use of the proxy causes the Ice run time to obtain another connection. Whether that connection uses a different endpoint than previous connections depends on a number of factors, but it is possible for the client to connect to a different server than for previous requests.

Locator Cache

After successfully resolving an indirect proxy, the location service must return at least one endpoint. How the service derives the list of endpoints that corresponds to the proxy is entirely implementation dependent. For example, IceGrid's location service can be configured to respond in a variety of ways; one possibility uses a simple round-robin scheme, while another selects endpoints based on the system load of the target hosts.

A locate request has the potential to significantly increase the latency of the application's invocation with a proxy, and this is especially true if the locate request triggers additional implicit actions such as starting a new server process. Fortunately, this overhead is normally incurred only during the application's initial invocation on the proxy, but this impact is influenced by the Ice run time's caching behavior.

To minimize the number of locate requests, the Ice run time caches the results of previous requests. By default, the results are cached indefinitely, so that once the Ice run time has obtained the endpoints associated with an indirect proxy, it never issues another locate request for that proxy. Furthermore, the default behavior of a proxy is to cache its connection, that is, once a proxy has obtained a connection, it continues to use that connection indefinitely.

Taken together, these two caching characteristics represent the Ice run time's best efforts to optimize an application's use of a location service: after a proxy is associated with a connection, all future invocations on that proxy are sent on the same connection without any need for cache lookups, locate requests, or new connections.

If a proxy's connection is closed, the next invocation on the proxy prompts the Ice run time to consult its locator cache to obtain the endpoints from the prior locate request. Next, the Ice run time searches for an existing connection to any of those endpoints and uses that if possible, otherwise it attempts to establish a new connection to each of the endpoints until one succeeds. Only if that process fails does the Ice run time clear the entry from its cache and issue a new locate request with the expectation that a usable endpoint is returned.

The Ice run time's default behavior is optimized for applications that require minimal interaction with the location service, but some applications can benefit from more frequent locate requests. Normally this is desirable when implementing a load-balancing strategy, as we discuss in more detail below. In order to increase the frequency of locate requests, an application must configure a timeout for the locator cache and manipulate the connections of its proxies.

Locator Cache Timeout

An application can define a timeout to control the lifetime of entries in the locator cache. This timeout can be specified globally using the `Ice.Default.LocatorCacheTimeout` property and for individual proxies using the proxy method `ice_locatorCacheTimeout`. The Ice run time's default behavior is equivalent to a timeout value of `-1`, meaning the cache entries never expire. Using a timeout value greater than zero causes the cache entries to expire after the specified number of seconds. Finally, a timeout value of zero disables the locator cache altogether.

The previous section explained the circumstances in which the Ice run time consults the locator cache. Briefly, this occurs only when the application has invoked an operation on a proxy and the proxy is not currently associated with a connection. If the timeout is set to zero, the Ice run time issues a new locate request immediately. Otherwise, for a non-zero timeout, the Ice run time examines the locator cache to determine whether the endpoints from the previous locate request have expired. If so, the Ice run time discards them and issues a new locate request.

Given this behavior, if your goal is to force a proxy invocation to issue locate requests more frequently, you can do so only when the proxy is not associated with a connection. You can accomplish that in several ways:

- create a new proxy, which is inherently not connected by default
- explicitly close the proxy's existing connection
- disabling the proxy's connection caching behavior

Of these choices, the last is the most common.

Proxy Connection Caching

By default a proxy remembers its connection and uses it for all invocations until that connection is closed. You can prevent a proxy from caching its connection by calling the `ice_connectionCached` proxy method with an argument of false. Once connection caching is disabled, each invocation on a proxy causes the Ice run time to execute its connection establishment process.

Note that each invocation on such a proxy does not necessarily cause the Ice run time to establish a new connection. It only means that the Ice run time does not assume that it can reuse the connection of the proxy's previous invocation. Whether the Ice run time actually needs to establish a new connection for the next invocation depends on several factors.

As with any feature, you should only use it when the benefits outweigh the risks. With respect to a proxy's connection caching behavior, there is certainly a small amount of computational overhead associated with executing the connection establishment process for each invocation, as well as the risk of significant overhead each time a new connection is actually created.

Simple Load Balancing

Several forms of load balancing are available to Ice applications. The simplest form uses only the endpoints contained in a direct proxy and does not require a location service. In this configuration, the application can configure the proxy to use the desired endpoint selection type and connection caching behavior to achieve the desired results.

For example, suppose that a proxy contains several endpoints. In its default configuration, it uses the `Random` endpoint selection type and caches its connection. Upon the first invocation, the Ice run time selects one of the proxy's endpoints at random and uses that connection for all subsequent invocations until the connection is closed. For some applications, this form of load balancing may be sufficient.

Suppose now that we use the `Ordered` endpoint selection type instead. In this case, the Ice run time always attempts to establish connections using the endpoints in the order they appear in the proxy. Normally an application uses this configuration when there is a preferred order to the servers. Again, once connected, the application uses whichever connection was chosen indefinitely.

By disabling the proxy's connection caching behavior, the semantics undergo a significant change. Using the `Random` endpoint selection type, the Ice run time selects one of the endpoints at random and establishes a connection to it if one is not already established, and this process is repeated prior to each subsequent invocation. This is called *per-request load balancing* because each request can be directed to a different server. Using the `Ordered` endpoint selection type is not as common in this scenario; its main purpose would be to fall back on a secondary server if the primary server is not available, but it causes the Ice run time to attempt to contact the primary server during each request.

Load Balancing with a Locator

A disadvantage of relying solely on the simple form of load balancing described above is that the client cannot make any intelligent decisions based on the status of the servers. If you want to distribute your requests in a more sophisticated way, you must either modify your clients to query the servers directly, or use a location service that can transparently direct a client to an appropriate server. For example, the IceGrid location service can monitor the system load on each server host and use that information when responding to locate requests.

The location service may return only one endpoint, which presumably represents the best server (at that moment) for the client to use. With only one endpoint available, changing the proxy's endpoint selection type makes no difference. However, by disabling connection caching and modifying the locator cache timeout, the application can force the Ice run time to periodically retrieve an updated endpoint from the location service. For example, an application can set a locator cache timeout of thirty seconds and communicate with the selected server for that period. After the timeout has expired, the next invocation prompts the Ice run time to issue a new locate request, at which point the client might be directed to a different server.

If the location service returns multiple endpoints, the application must be designed with knowledge of how to interpret them. For instance, the location service may attach semantics to the order of the endpoints (such as least-loaded to most-loaded) and intend that the application use the endpoints in the order provided. Alternatively, the client may be free to select any of the endpoints. As a result, the application and the location service must cooperate to achieve the desired results.

You can combine the simple form of load balancing described in the previous section with an intelligent location service to gain even more flexibility. For example, suppose an application expects to receive multiple endpoints from the location service and has configured its proxy to disable connection caching and set a locator cache timeout. For each invocation, Ice run time selects one of the endpoints provided by the location service. When the timeout expires, the Ice run time issues a new locate request and obtains a fresh set of endpoints from which to choose.

See Also

- Terminology
- Proxy Methods
- Connection Establishment
- Active Connection Management
- Ice Default and Override Properties