Connection Establishment

Connections are established as a side effect of using proxies. The first invocation on a proxy causes the Ice run time to search for an existing connection to one of the proxy's endpoints; only if no suitable connection exists does the Ice run time establish a new connection to one of the proxy's endpoints.

This page describes how and when Ice establishes a new connection.

On this page:

- Endpoint Selection for New Connections
- Error Semantics for Failed Connections
- Reusing an Existing Connection
  - Connection Reuse for Proxies with Multiple Endpoints
  - Protocol Compression and Connection Reuse
  - Influencing Connection Reuse
- Connection Caching
- Timeouts and Connection Establishment

Endpoint Selection for New Connections

A proxy performs a number of operations on its endpoints before it asks the Ice run time to supply a connection. These operations produce a list of zero or more endpoints that satisfy the proxy's configuration. If the resulting list is empty, the application receives `NoEndpointException` to indicate that no suitable endpoints could be found. For example, this situation can arise when a twoway proxy contains only a UDP endpoint; the UDP endpoint is eliminated from consideration because it cannot be used for twoway invocations.

The proxy performs the following steps to derive its endpoint list:

1. Remove the endpoints of unknown transports. For instance, SSL endpoints are removed if the SSL plug-in is not installed.
2. Remove endpoints that are not suitable for the proxy's invocation mode. For example, datagram endpoints are removed for twoway, oneway and batch oneway proxies. Similarly, non-datagram endpoints are removed for datagram and batch datagram proxies.
3. Perform DNS queries to convert host names into IP addresses, if necessary. For a multi-homed host name, the proxy adds a new endpoint for each address returned by the DNS query.
4. Sort the endpoints according to the configured selection type, which is established using the `ice_endpointSelection` proxy method. The default value is `Random`, meaning the endpoints are randomly shuffled. Alternatively, the value `Ordered` maintains the existing order of the endpoints.
5. Satisfy the proxy's security requirements:
   - If `Ice.Override.Secure` is defined, remove all non-secure endpoints.
   - Otherwise, if the proxy is configured to prefer secure endpoints (e.g., by calling the `ice_preferSecure` proxy method), move all secure endpoints to the beginning of the list. Note that this setting still allows non-secure endpoints to be included.
   - Otherwise, move all non-secure endpoints to the beginning of the list.

If connection caching is enabled and the Ice run time already has a compatible connection, it reuses the cached connection. Otherwise, the run time attempts to connect to each endpoint in the list until it succeeds or exhausts the list; the order in which endpoints are selected for connection attempts depends on the endpoint selection policy. This policy can be set using a default property (`Ice.Default.EndpointSelection`), using a proxy property (*`name`*`.EndpointSelection`), and using the `ice_endpointSelection` proxy method.

Error Semantics for Failed Connections

If a failure occurs during a connection attempt, the Ice run time tries to connect to all of the proxy's remaining endpoints until either a connection is successfully established or all attempts have failed. At that point, the Ice run time may attempt automatic retries depending on the value of the `Ice.RetryIntervals` configuration property. The default value of this property is `0`, which causes the Ice run time to try connecting to all of the endpoints one more time.

Tip

Define the property `Ice.Trace.Retry=2` to monitor these attempts.

If no connection can be established on this second attempt, the Ice run time raises an exception that indicates the reason for the final failed attempt (typically `ConnectFailedException`). Similarly, if a connection was lost during a request and could not be reestablished (assuming the request can be retried), the Ice run time raises an exception that indicates the reason for the final failed attempt.

Copyright © 2017, ZeroC, Inc.

Reusing an Existing Connection

When establishing a connection for a proxy, the Ice run time reuses an existing connection under the following conditions:

- The remote endpoint matches one of the proxy's endpoints.
- The connection was established by the communicator that created the proxy.
- The connection matches the proxy's configuration. Timeout values play an important role here, as an existing connection is only reused if its timeout value (i.e., the timeout used when the connection was established) matches the new proxy's timeout. Similarly, a proxy configured with a connection ID only reuses a connection if it was established by a proxy with the same connection ID.

Connection Reuse for Proxies with Multiple Endpoints

Applications must exercise caution when using proxies containing multiple endpoints, especially endpoints using different transports. For example, suppose a proxy has multiple endpoints, such as one each for TCP, SSL, and UDP. When establishing a connection for this proxy, the Ice run time will open a new connection only if it cannot reuse an existing connection to any of the endpoints (that is, if connection caching is enabled). Furthermore, the proxy in its default (that is, non-secure) configuration gives higher priority to non-secure endpoints. If you want to ensure that a particular transport is used by a proxy, you must configure the proxy appropriately, such as by calling the proxy methods `ice_secure` or `ice_datagram` as necessary.

Protocol Compression and Connection Reuse

The Ice run time does not consider compression settings when searching for existing connections to reuse; proxies whose compression settings differ can share the same connection (assuming all other selection criteria are satisfied).

Influencing Connection Reuse

The default behavior of the Ice run time, which reuses connections whenever possible, is appropriate for many applications because it conserves resources and typically has little or no impact on performance. However, when a server implementation attaches semantics to a connection, the client often must be designed to cooperate, despite the tighter coupling it causes. For example, a server might use a serialized thread pool to preserve the order of requests received over each connection. If the client wants to execute several requests simultaneously, it must be able to force the Ice run time to establish new connections at will.

For those situations that require more control over connection reuse, the Ice run time allows you to form arbitrary groups of proxies that share a connection by configuring them with the same connection identifier. The proxy method `ice_connectionId` returns a new proxy configured with the given connection ID. Once configured, the Ice run time ensures that the proxy only reuses a connection that was established by a proxy with the same connection ID (assuming all other criteria for connection reuse are also satisfied). A new connection is created if none with a matching ID is found, which means each proxy could conceivably have its own connection if each were assigned a unique connection ID.

As an example, consider the following code fragment:

C++

```
Ice::ObjectPrx prx = comm->stringToProxy("ident:tcp -p 10000");
Ice::ObjectPrx g1 = prx->ice_connectionId("group1");
Ice::ObjectPrx g2 = prx->ice_connectionId("group2");
prx->ice_ping(); // Opens a new connection
g1->ice_ping(); // Opens a new connection
g2->ice_ping(); // Opens a new connection
MyInterfacePrx i1 = MyInterfacePrx::checkedCast(g1);
i1->ice_ping(); // Reuses g1's connection
MyInterfacePrx i2 = MyInterfacePrx::checkedCast(prx->ice_connectionId("group2"));
i2->ice_ping(); // Reuses g2's connection
```

A total of three connections are established by this example:

1. The proxy `prx` establishes a new connection. This proxy has the default connection ID (an empty string).
2. The proxy `g1` establishes a new connection because the only existing connection, the one established by `prx`, has a different connection ID.
3. Similarly, the proxy `g2` establishes a new connection because none of the existing connections have a matching connection ID.

The proxy `i1` inherits its connection ID from `g1`, and therefore shares the connection for `group1`; `i2` explicitly configured its connection ID and shares the `group2` connection with proxy `g2`.

Copyright © 2017, ZeroC, Inc.

Connection Caching

When we refer to a proxy's connection, we actually mean the connection that the proxy is *currently* using. This connection can change over time, such that a proxy might use several connections during its lifetime. For example, an idle connection may be closed automatically and then transparently replaced by a new connection when activity resumes.

After establishing a connection in response to proxy activities, the Ice run time adds the connection to an internal pool for subsequent reuse by other proxies. The Ice run time manages the lifetime of the connection and eventually closes it. The connection is not affected by the life cycle of the proxies that use it, except that the lack of activity may prompt the Ice run time to close the connection after a while.

Once a proxy has been associated with a connection, the proxy's default behavior is to continue using that connection for all subsequent requests. In effect, the proxy caches the connection and attempts to use it for as long as possible in order to minimize the overhead of creating new connections. If the connection is later closed and the proxy is used again, the proxy repeats the connection-establishment procedure described earlier.

There are situations in which this default caching behavior is undesirable, such as when a client has a proxy with multiple endpoints and wishes to balance the load among the servers at those endpoints. The client can disable connection caching by passing an argument of `false` to the proxy method `ice_connectionCached`. The new proxy returned by this method repeats the connection-establishment procedure before each request, thereby achieving request load balancing at the expense of potentially higher latency.

This type of load balancing is performed solely by the client using whatever endpoints are contained in the proxy. More sophisticated forms of load balancing are also possible, such as when using IceGrid.

Timeouts and Connection Establishment

A proxy's default configuration has a timeout value of `-1`, meaning that network activity initiated by this proxy does not time out. The timeout value affects both connection establishment and remote invocations. If a different timeout value is specified and the connection cannot be established within the allotted time, a `ConnectTimeoutException` is raised.

You can set a timeout on a proxy using the `ice_timeout` proxy method. To use the same timeout period for all proxies, you can define the `Ice.Override.Timeout` property; in this case, any timeout established using the `ice_timeout` proxy method is ignored. Finally, if you want to specify a separate timeout value that affects only connection establishment and takes precedence over a proxy's configured timeout value, you can define the `Ice.Override.ConnectTimeout` property.

The timeout in effect when a connection is established is bound to that connection and affects all requests on that connection. If a request times out, all other outstanding requests on the same connection also time out, and the connection is closed forcefully. The Ice run time automatically retries these requests on a new connection, assuming that automatic retries are enabled and would not violate at-most-once semantics.

See Also

- Proxy Methods
- Proxy Endpoints
- The Ice Threading Model
- Automatic Retries
- Active Connection Management
- IceGrid
- Ice Default and Override Properties
- Ice Proxy Properties
- Ice Miscellaneous Properties

Copyright © 2017, ZeroC, Inc.