Routers

Previous Next

This page describes the Ice router facility.

On this page:

Router Overview

A *router* is an Ice object that provides information to the Ice run time to allow routing messages between clients and servers. In a client, configuring a proxy to use a router produces a *routed proxy* on which all invocations are sent to the router for forwarding to the target server. The endpoints in a routed proxy are ignored by the Ice run time, at least for the purposes of connection establishment. Instead, the Ice run time in the client establishes a connection to the router and may pass along the proxy's endpoints so that the router can establish its own connection to the target server. Ice typically opens only one connection to a router and reuses that connection for invocations on all of the routed proxies configured to use the same router.

The Ice distribution includes two router implementations that serve different purposes:

- Glacier2 is a sophisticated routing service that is commonly used as a single point of entry through which clients on public networks access back-end services on an internal network, with support for application-specific session management and access control.
- IceBridge is a simpler service that facilitates bridging connections over different transports.

Both services implement the `Ice::Router` interface, which the Ice run time requires of any router implementation.

Slice

```
module Ice
{
    interface Router
    {
        ["nonmutating", "cpp:const"] idempotent Object* getClientProxy(out optional(1) bool hasRoutingTable);

        ["nonmutating", "cpp:const"] idempotent Object* getServerProxy();

        idempotent ObjectProxySeq addProxies(ObjectProxySeq proxies);
    }

    interface RouterFinder
    {
        Router* getRouter();
    }
}
```

Back to Top ^

Default Router

A communicator can be configured with a default router. The most common way to configure the default router is to set the property `Ice.Default.Router`. The value of this property is a proxy for the router's primary Ice object, as shown in the example below for Glacier2:

```
Ice.Default.Router=Glacier2/router:tcp -h routerhost -p 4063
```

You can also specify a default router by calling `setDefaultRouter` on the communicator, and obtain the current setting using `getDefaultRouter`.

Back to Top ^

Configuring a Router for Client Invocations

Setting a default router as described above means every proxy created by the communicator will be configured to use the router by default. If your client needs to use a router more selectively, you can use the `ice_router` proxy method to obtain a routed proxy:

C++

```
shared_ptr<Ice::RouterPrx> router = ...;
auto target = communicator->stringToProxy("...");
target = target->ice_router(router);
```

As with all proxy factory methods, `ice_router` returns a new proxy with the requested configuration.

Another way to configure a router is with a proxy property:

```
MyProxy.Router=Glacier2/router:tcp -h routerhost -p 4063
```

In this example, calling `propertyToProxy("MyProxy")` on a communicator returns a proxy that is already configured to use Glacier2.

Back to Top ^

Configuring a Router for Callback Invocations

A client that needs to receive callback invocations may need to perform some additional configuration, depending on the router implementation. For example, a Glacier2 client needs to configure its object adapter so that its callback proxies contain the appropriate endpoints. On the other hand, with IceBridge you simply configure a bidirectional connection.

Back to Top ^

Routing Tables

A router implementation may optionally maintain an internal routing table that the Ice client run time populates automatically by calling `addProxies`.

For example, Glacier2 uses a routing table because it can forward requests to any number of back-end servers, whereas IceBridge does not use a routing table because each IceBridge instance is statically configured to forward requests to a single server. When a client makes an initial invocation on a routed proxy, and the configured router uses a routing table, the Ice run time in the client needs to send that routed proxy to the router so that the router has the endpoint information necessary to establish a connection to the back-end server. The Ice run time in the client also maintains its own local version of the routing table in order to minimize overhead; Ice only sends each routed proxy to the router once. Furthermore, Ice keeps its table synchronized with the router's by tracking any proxies that the router might have evicted from its table (the proxies returned by the call to `addProxies`).

Ice uses object identities as the keys in its routing table, which means it's important that your Ice objects use unique identities.

If a router receives an invocation to be forwarded for an object identity that is not in its routing table (e.g., it may have been recently evicted), the router raises `Ice::ObjectNotExistException` and sets the `operation` member to the reserved value `ice_add_proxy` in order to notify the Ice run time in the client that the routed proxy is unknown. The Ice client run time must therefore register the proxy with the router and retry the invocation.

Back to Top ^

See Also

- Glacier2
- IceBridge

Previous

Next