Glacier2 Session Management

A Glacier2 router requires a client to create a session and forwards requests on behalf of the client until its session expires. A session expires when it is explicitly destroyed, or when it times out due to inactivity.

You can configure a router to use a custom session manager if your application needs to track the router's session activities. For example, your application may need to acquire resources and initialize the state of back-end services for each new session, and later reclaim those resources when the session expires.

As with the authentication facility, Glacier2 provides two session manager interfaces that an application can implement. The `SessionManager` interface receives notifications about sessions that use password authentication, while the `SSLSessionManager` interface is for sessions authenticated using SSL certificates.

On this page:

Glacier2 Session Manager Interfaces

The relevant Slice definitions are shown below:

Slice

```
module Glacier2 {
    exception CannotCreateSessionException {
        string reason;
    };

    interface Session {
        void destroy();
    };

    interface SessionManager {
        Session* create(string userId, SessionControl* control)
            throws CannotCreateSessionException;
    };

    interface SSLSessionManager {
        Session* create(SSLInfo info, SessionControl* control)
            throws CannotCreateSessionException;
    };
};
```

When a client creates a session by invoking `createSession` on the `Router` interface, the router validates the client's user name and password and then calls `SessionManager::create`. Similarly, a call to `createSessionFromSecureConnection` causes the router to invoke `SSLSessionManager::create`. The `SSLInfo` structure provides details about the client's SSL connection. The second argument to the `create` operations is a proxy for a `SessionControl` object, which a session can use to perform dynamic filtering.

The `create` operations must return the proxy of a new `Session` object, or raise `CannotCreateSessionException` and provide an appropriate reason. The `Session` proxy returned by `create` is ultimately returned to the client as the result of `createSession` or `createSessionFromSecureConnection`.

Glacier2 invokes the `destroy` operation on a `Session` proxy when the session expires, giving a custom session manager the opportunity to reclaim resources that were acquired for the session during `create`.

ⓘ The `create` operations may be called with information that identifies an existing session. For example, this can occur when a client loses its connection to the router but its previous session has not yet expired (and therefore the router has not yet invoked `destroy` on its `Session` proxy). A session manager implementation must be prepared to handle this situation.

To configure the router with a custom session manager, define the properties `Glacier2.SessionManager` or `Glacier2.SSLSessionManager` with the proxies of the session manager objects. If necessary, you can configure a router with proxies for both types of session managers. If a session manager proxy is not supplied, the call to `createSession` or `createSessionFromSecureConnection` always returns a null proxy.

The router attempts to contact the configured session manager at startup. If the object is unreachable, the router logs a warning message but continues its normal operation (you can suppress the warning using the `--nowarn` option). The router does not contact the session manager again until it needs to invoke an operation on the object. For example, when a client asks the router to create a new session, the router makes another attempt to contact the session manager; if the session manager is still unavailable, the router logs a message and returns `CannotCreateSessionException` to the client.

Example

A sample implementation of the `SessionManager` interface is provided in the `demo/Glacier2/callback` directory.

Glacier2 Session Timeouts

The value of the `Glacier2.SessionTimeout` property specifies the number of seconds a session must be inactive before it expires. This property is not defined by default, which means sessions never expire due to inactivity. If a non-zero value is specified, it is very important that the application chooses a value that does not result in premature session expiration. For example, if it is normal for a client to create a session and then have long periods of inactivity, then a suitably long timeout must be chosen, or the client must actively keep its session alive, or timeouts must be disabled altogether.

Once a session has expired (or been destroyed for some other reason), the client will no longer be able to send requests via the router and receive a `ConnectionLostException` instead. The client must explicitly create a new session in order to continue using the router. If necessary, a client can use a dedicated thread to keep its session alive.

In general, we recommend the use of an appropriate session timeout, otherwise resources created for each session will accumulate in the router. However, you can safely disable the session timeout if the server regularly calls back to the client. In that case, Glacier2 will automatically destroy the session if a failure occurs while forwarding a server callback to the client.

Invocation Timeouts on Routed Proxies

If your client requires invocation timeouts for routed proxies, you must set the timeout on the router proxy that you use to establish the session. This is because the Ice run time forwards the invocation to Glacier2, and the timeout applies to that invocation.

In other words, whatever timeout you set on the router proxy that you use to create the session is the timeout that applies to all routed proxies. Do not attempt to override the timeout on a per-proxy basis; if you do, any setting other than the timeout used to establish the session results in a `ConnectionLostException`. This is because proxies with different timeout values establish separate connections, but there can be only one connection to Glacier2.

A future version of Ice may make it illegal to set a timeout on a routed proxy.

For invocations made by Glacier2 to the server, whatever timeout value is set on the *first* proxy that is used to make an invocation applies to all proxies for the same object. This is because Glacier2 adds the proxy to its routing table during the first invocation and, thereafter, reuses that cached proxy for all invocations to the same object identity. Here is an example to illustrate this:

C++

```cpp
// 10-second session timeout for router.
ObjectPrx router = communicator->stringToProxy("Glacier2/router:tcp -h host1 -p 4063 -t 10000");
communicator->setDefaultRouter(RouterPrx::uncheckedCast(router));

// Ping with 20-second timeout
communicator->stringToProxy("id:tcp -h host2 -p 12345 -t 20000")->ice_ping();

// Ping with 30-second timeout
communicator->stringToProxy("id:tcp -h host2 -p 12345 -t 30000")->ice_ping();
```

In this case, all invocations made by the client use a 10-second timeout to forward the invocations to Glacier2. The first call to `ice_ping`, when forwarded by Glacier2 to the server, uses a 20-second timeout. The second call to `ice_ping` also uses a 20-second timeout, even though the proxy specifies a 30-second timeout.

If you have a timeout on both the client-Glacier2 and the Glacier2-server connections, the timeout on the client-Glacier2 connection should be slightly longer; otherwise, invocation timeouts that are encountered by Glacier2 when it forwards an operation to the server cannot be propagated back to the client.

Connection Caching for Session Managers

You can distribute the load among multiple session manager objects by configuring the router with a session manager proxy that contains multiple endpoints. Glacier2 disables connection caching on this proxy so that each invocation on a session manager attempts to use a different endpoint.

This behavior achieves a basic form of load balancing without depending on the replication features provided by IceGrid. Be aware that including an invalid endpoint in your session manager proxy, such as the endpoint of a session manager server that is not currently running, can cause router clients to experience delays during session creation.

If your session managers are in an IceGrid replica group, refer to IceGrid and Glacier2 Integration for more information on the router's caching behavior.

See Also

- Getting Started with Glacier2
- Securing a Glacier2 Router
- Dynamic Request Filtering with Glacier2
- IceGrid and Glacier2 Integration
- Object Adapter Replication
- Glacier2 Properties