Dynamic Request Filtering with Glacier2

Glacier2 can be statically configured to filter requests, and also allows a session manager to customize filters for each session at run time via its `SessionControl` interface:

Slice

```
module Glacier2 {
    interface SessionControl {
        StringSet* categories();
        StringSet* adapterIds();
        IdentitySet* identities();
        void destroy();
    };
};
```

The router creates a `SessionControl` object for each client session and supplies a proxy for the object to the session manager `create` operations. Note that the `SessionControl` proxy is null unless the router is configured with server endpoints.

Invoking the `destroy` operation causes the router to destroy the client's session, which eventually results in an invocation of `destroy` on the application-defined `Session` object, if one was provided.

The interface operations `categories`, `adapterIds`, and `identities` return proxies to objects representing the modifiable filters for the session. The router initializes these filters using their respective static configuration properties.

The `SessionControl` object uses a `StringSet` to manage the category and adapter identifier filters:

Slice

```
module Glacier2 {
    interface StringSet {
        idempotent void add(Ice::StringSeq additions);
        idempotent void remove(Ice::StringSeq deletions);
        idempotent Ice::StringSeq get();
    };
};
```

Similarly, the `IdentitySet` interface manages the identity filters:

Slice

```
module Glacier2 {
    interface IdentitySet {
        idempotent void add(Ice::IdentitySeq additions);
        idempotent void remove(Ice::IdentitySeq deletions);
        idempotent Ice::IdentitySeq get();
    };
};
```

In both interfaces, the `add` operation silently ignores duplicates, and the `remove` operation silently ignores non-existent entries.

Dynamic filtering is often necessary when each session must be restricted to a particular group of objects. Upon session creation, a session manager typically allocates a number of objects in back-end servers for that session to use. To prevent other sessions from accessing these objects (intentionally or not), the session manager can configure the session's filters so that it is only permitted to use the objects that were created for it.

For example, a session manager can retain the `SessionControl` proxy and add a new identity to the `IdentitySet` as each new object is created for the session. A simpler solution is to create a unique category for the session, add it to the session's category filter, and use that category in the identities of all objects accessible to that session. Using a category filter in this way reserves an identity namespace for each session and avoids the need to update the filter for each new object.

To aid in logging and debugging, you can select a category that identifies the client, such as the user name that was supplied during session creation, or an attribute of the client's certificate such as the common name, as long as the selected category is sufficiently unique that it will not conflict with another client's session. You must also ensure that the categories you assign to sessions never match the categories of back-end objects that are not meant to be accessed by router clients. As an example, consider the following session manager implementation:

C++

```
class SessionManagerI : public Glacier2::SessionManager
{
public:

    virtual Glacier2::SessionPrx
    create(const string& username, const Glacier2::SessionControlPrx& ctrl,
           const Ice::Current& curr)
    {
        string category = "_" + username;
        ctrl->categories()->add(category);
        // ...
    }
};
```

This session manager derives a category for the session by prepending an underscore to the user name and then adds this category to the session's filter. As long as our back-end objects do not use a leading underscore in their identity categories, this strategy guarantees that a session's category can never match the category of a back-end object.

For your convenience, Glacier2 already includes support for automatic category filtering.

See Also

- Callbacks through Glacier2
- Securing a Glacier2 Router
- Glacier2 Session Management