Connection Timeouts

A synchronous remote invocation does not complete on the client side until the server has finished processing it. Occasionally, it is useful to be able to force an invocation to terminate after some time, even if it has not completed. The `ice_timeout` factory method is provided for this purpose. The Slice definition of `ice_timeout` would look as follows:

Slice

```
Object* ice_timeout(int t);
```

This method returns a new proxy with the configured timeout. For example:

C++

```
Filesystem::FilePrx myFile = ...;
FileSystem::FilePrx timeoutFile = myFile->ice_timeout(5000);

try {
    Lines text = timeoutFile->read();   // Read with timeout
} catch(const Ice::TimeoutException&) {
    cerr << "invocation timed out" << endl;
}

Lines text = myFile->read();            // Read without timeout
```

The parameter to `ice_timeout` determines the timeout value in milliseconds. A value of `-1` indicates no timeout. In the preceding example, the timeout is set to five seconds; if an invocation of `read` via the `timeoutFile` proxy does not complete within five seconds, the operation terminates with an `Ice::TimeoutException`. On the other hand, invocations via the `myFile` proxy are unaffected by the timeout, that is, `ice_timeout` sets the timeout on a per-proxy basis.

The timeout value set on a proxy affects all networking operations: reading and writing of data as well as opening and closing of connections. If any of these operations does not complete within the timeout, the client receives an exception. Note that, if the Ice run time encounters a recoverable error condition and transparently retries an invocation, this means that the timeout applies separately to each attempt. Similarly, if a large amount of data is sent with an operation invocation in several `write` system calls, the timeout applies to each write, not to the invocation overall.

Timeouts that expire during reading or writing of data are indicated by a `TimeoutException`. For opening and closing of connections, the Ice run time uses more specific exceptions:

- `ConnectTimeoutException`
  This exception indicates that a connection could not be established within the specified time.
- `CloseTimeoutException`
  This exception indicates that a connection could not be closed within the specified time.

An application normally configures a proxy's timeout using the `ice_timeout` method. However, a proxy that originated from a string may already have a timeout specified, as shown in the following example:

C++

```
string s = "ident:tcp -h somehost -t 5000:ssl -h somehost -t 5000";
```

In this case, both the TCP and SSL endpoints define a timeout of five seconds. When the Ice run time establishes a connection using one of these endpoints, it uses the endpoint's timeout unless one was specified explicitly via `ice_timeout`.

The Ice run time also supports two configuration properties that override the timeouts of every proxy regardless of the settings established via `ice_timeout` or the options defined in stringified proxies:

- `Ice.Override.ConnectTimeout`
  This property defines a timeout that is only used for connection establishment. If not defined, its default value is `-1` (no timeout). If a proxy has multiple endpoints, the timeout applies to each endpoint separately.
- `Ice.Override.Timeout`
  This property defines the timeout for invocations. If no value is defined for `Ice.Override.ConnectTimeout`, the value of `Ice.Override.Timeout` is also used as the timeout for connection establishment. If not defined, the default value is `-1` (no timeout).

If you are considering the use of timeouts, please keep the following limitations in mind:

- Timeouts are "soft" timeouts, in the sense that they are not precise, real-time timeouts. (The precision is limited by the capabilities of the underlying operating system.)
- Timeouts are considered fatal error conditions by the Ice run time and result in connection closure on the client side. Furthermore, any other requests pending on the same connection also fail with an exception.
- Timeouts are meant to be used to prevent a client from blocking indefinitely in case something has gone wrong with the server; they are not meant as a mechanism to routinely abort requests that take longer than intended.
- Be careful when using timeouts with Glacier2.

See Also

- Proxy Methods
- Connection Establishment