Threading Guarantees for Servant Locators

The Ice run time guarantees that every operation invocation that involves a servant locator is bracketed by calls to `locate` and `finished`, that is, every call to `locate` is balanced by a corresponding call to `finished` (assuming that the call to `locate` actually returned a servant, of course).

In addition, the Ice run time guarantees that `locate`, the operation, and `finished` are called by the same thread. This guarantee is important because it allows you to use `locate` and `finished` to implement thread-specific pre- and post-processing around operation invocations. (For example, you can start a transaction in `locate` and commit or roll back that transaction in `finished`, or you can acquire a lock in `locate` and release the lock in `finished`.

Both transactions and locks usually are thread-specific, that is, only the thread that started a transaction can commit it or roll it back, and only the thread that acquired a lock can release the lock.

If you are using asynchronous method dispatch, the thread that starts a call is not necessarily the thread that finishes it. In that case, `finished` is called by whatever thread executes the operation implementation, which may be a different thread than the one that called `locate`.

The Ice run time also guarantees that `deactivate` is called when you destroy the object adapter to which the servant locator is attached. The `deactivate` call is made only once all operations that involved the servant locator are finished, that is, `deactivate` is guaranteed not to run concurrently with `locate` or `finished`, and is guaranteed to be the last call made to a servant locator.

Beyond this, the Ice run time provides no threading guarantees for servant locators. In particular, it is possible for invocations of:

- `locate` to proceed concurrently (for the same object identity or for different object identities).
- `finished` to proceed concurrently (for the same object identity or for different object identities).
- `locate` and `finished` to proceed concurrently (for the same object identity or for different object identities).

These semantics allow you to extract the maximum amount of parallelism from your application code (because the Ice run time does not serialize invocations when serialization may not be necessary). Of course, this means that you must protect access to shared data from `locate` and `finishe`
with mutual exclusion primitives as necessary.

See Also

- The ServantLocator Interface
- The Ice Threading Model