Object Adapter States

On this page:

- Object Adapter State Transitions
- Changing Object Adapter States

Object Adapter State Transitions

An object adapter has a number of processing states:

- **Holding**
  In this state, any incoming requests for the adapter are held, that is, not dispatched to servants.

  For TCP/IP (and other stream-oriented protocols), the server-side run time stops reading from the corresponding transport endpoint while the adapter is in the holding state. In addition, it also does not accept incoming connection requests from clients. This means that if a client sends a request to an adapter that is in the holding state, the client eventually receives a `TimeoutException` or `ConnectTimeoutException` (unless the adapter is placed into the active state before the timer expires).

  For UDP, client requests that arrive at an adapter that is in the holding state are thrown away.

  Immediately after creation of an adapter, the adapter is in the holding state. This means that requests are not dispatched until you place the adapter into the active state.

- **Active**
  In this state, the adapter accepts incoming requests and dispatches them to servants. A newly-created adapter is initially in the holding state. The adapter begins dispatching requests as soon as you place it into the active state.

  You can transition between the active and the holding state as many times as you wish.

- **Inactive**
  In this state, the adapter has conceptually been destroyed (or is in the process of being destroyed). Deactivating an adapter destroys all transport endpoints that are associated with the adapter. Requests that are executing at the time the adapter is placed into the inactive state are allowed to complete, but no new requests are accepted. (New requests are rejected with an exception). Any attempt to use a deactivated object adapter results in an `ObjectAdapterDeactivatedException`.



The distinction `Holding` vs Active `applies` only to requests dispatched through endpoints configured on the object adapter. It does not apply to requests received from bidirectional connections, nor does it apply to collocated dispatches (unless you disable collocation optimization).

An object adapter configured with a router accepts only requests received over the (bidirectional) connection to the router, and collocated dispatches.

Changing Object Adapter States

The `ObjectAdapter` interface offers operations that allow you to change the adapter state, as well as to wait for a state change to be complete:

Copyright 2018, ZeroC, Inc.

Slice

```
module Ice
{
    local interface ObjectAdapter
    {
        // ...

        void activate();
        void hold();
        void waitForHold();
        void deactivate();
        void waitForDeactivate();
        void isDeactivated();
        void destroy();

        // ...
    }
}
```

The operations behave as follows:

- `activate`
  The `activate` operation places the adapter into the active state. Activating an adapter that is already active has no effect. The Ice run time starts dispatching requests to servants for the adapter as soon as `activate` is called.

- `hold`
  The `hold` operation places the adapter into the holding state. Requests that arrive after calling `hold` are held as described above. Requests that are in progress at the time `hold` is called are allowed to complete normally. Note that `hold` returns immediately without waiting for currently executing requests to complete.

- `waitForHold`
  The `waitForHold` operation suspends the calling thread until the adapter has completed its transition to the holding state, that is, until all currently executing requests have finished. You can call `waitForHold` from multiple threads, and you can call `waitForHold` while the adapter is in the active state. If you call `waitForHold` on an adapter that is already in the holding state, `waitForHold` returns immediately.

- `deactivate`
  The `deactivate` operation initiates deactivation of the adapter: requests that arrive after calling `deactivate` are rejected, but currently executing requests are allowed to complete. Once all requests have completed, the adapter enters the inactive state. Note that `deactivate` returns immediately without waiting for the currently executing requests to complete. A deactivated adapter cannot be reactivated; you can create a new adapter with the same name, but only after calling `destroy` on the existing adapter. Any attempt to use a deactivated object adapter results in an `ObjectAdapterDeactivatedException`. During deactivation, the association with bidirectional connections (if any) is cleared. Deactivation also makes the adapter ineligible for collocated dispatches.

- `waitForDeactivate`
  The `waitForDeactivate` operation suspends the calling thread until the adapter has completed its transition to the inactive state, that is, until all currently executing requests have completed. You can call `waitForDeactivate` from multiple threads, and you can call `waitForDeactivate` while the adapter is in the active or holding state. Calling `waitForDeactivate` on an adapter that is in the inactive state does nothing and returns immediately.

- `isDeactivated`
  The `isDeactivated` operation returns true if `deactivate` has been invoked on the adapter. A return value of true does not necessarily indicate that the adapter has fully transitioned to the inactive state, only that it has begun this transition. Applications that need to know when deactivation is completed can use `waitForDeactivate`.

- `destroy`
  The `destroy` operation deactivates the adapter and releases all of its resources. Internally, `destroy` invokes `deactivate` followed by `waitForDeactivate`, therefore the operation blocks until all currently executing requests have completed. Furthermore, any servants associated with the adapter are destroyed, all transport endpoints are closed, and the adapter's name becomes available for reuse. Destroying a communicator implicitly destroys all of its object adapters. Invoking `destroy` on an adapter is only necessary when you need to ensure that its resources are released prior to the destruction of its communicator.

Placing an adapter into the holding state is useful, for example, if you need to make state changes in the server that require the server (or a group of servants) to be idle. For example, you could place the implementation of your servants into a dynamic library and upgrade the implementation by loading a newer version of the library at run time without having to shut down the server.

Similarly, waiting for an adapter to complete its transition to the inactive state is useful if your server needs to perform some final clean-up work that cannot be carried out until all executing requests have completed.

Copyright 2018, ZeroC, Inc.

Note that you can create an object adapter with the same name as a previous object adapter, but only once `destroy` on the previous adapter has completed.

See Also

- Collocated Invocation and Dispatch
- Bidirectional Connections

Copyright 2018, ZeroC, Inc.