Servant Activation and Deactivation

Previous Next

The term *servant activation* refers to making the presence of a servant for a particular Ice object known to the Ice run time. Activating a servant adds an entry to the Active Servant Map (ASM). Another way of looking at servant activation is to think of it as creating a link between the identity of an Ice object and the corresponding programming-language servant that handles requests for that Ice object. Once the Ice run time has knowledge of this link, it can dispatch incoming requests to the correct servant. Without this link, that is, without a corresponding entry in the ASM, an incoming request for the identity results in an `ObjectNotExistException`. While a servant is activated, it is said to *incarnate* the corresponding Ice object.

The inverse operation is known as *servant deactivation*. Deactivating a servant removes an entry for a particular identity from the ASM. Thereafter, incoming requests for that identity are no longer dispatched to the servant and result in an `ObjectNotExistException`.

The object adapter offers a number of operations for managing servant activation and deactivation:

Slice

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

        Object* add(Object servant, Identity id);
        Object* addWithUUID(Object servant);
        Object  remove(Identity id);
        Object  find(Identity id);
        Object  findByProxy(Object* proxy);

        // ...
    }
}
```

Back to Top ^

The operations behave as follows:

- `add`

  The `add` operation adds a servant with the given identity to the ASM. Requests are dispatched to that servant as soon as `add` is called. The return value is the proxy for the Ice object incarnated by that servant. The proxy embeds the identity passed to `add`.

  You cannot call `add` with the same identity more than once: attempts to add an already existing identity to the ASM result in an `AlreadyRegisteredException`. (It does not make sense to add two servants with the same identity because that would make it ambiguous as to which servant should handle incoming requests for that identity.)

  Note that it is possible to activate the same servant multiple times with different identities. In that case, the same single servant incarnates multiple Ice objects. We explore the ramifications of this in more detail in our discussion of server implementation techniques.

- `addWithUUID`

  The `addWithUUID` operation behaves the same way as the `add` operation but does not require you to supply an identity for the servant. Instead, `addWithUUID` generates a UUID as the identity for the corresponding Ice object. You can retrieve the generated identity by calling the `ice_getIdentity` operation on the returned proxy. `addWithUUID` is useful to create identities for temporary objects, such as short-lived session objects. (You can also use `addWithUUID` for persistent objects that do not have a natural identity, as we have done for the file system application.)

- `remove`

  The `remove` operation breaks the association between an identity and its servant by removing the corresponding entry from the ASM; it returns a reference to the removed servant.

  Once the servant is deactivated, new incoming requests for the removed identity cause the client to receive an `ObjectNotExistException`. Requests that are executing inside the servant at the time `remove` is called are allowed to complete normally. Once the last request for the servant is complete, the object adapter drops its reference (or smart pointer, for C++) to the servant. At that point, the servant becomes available for garbage collection (or is destroyed, for C++), provided there are no other references or smart pointers to the servant. The net effect is that a deactivated servant is destroyed once it becomes idle.

  Deactivating an object adapter implicitly calls `remove` on its active servants.

- `find`

The `find` operation performs a lookup in the ASM and returns the servant for the specified object identity. If no servant with that identity is registered, the operation returns null. Note that `find` does not consult any servant locators or default servants.

- `findByProxy`

  The `findByProxy` operation performs a lookup in the ASM and returns the servant with the object identity and facet that are embedded in the proxy. If no such servant is registered, the operation returns null. Note that `findByProxy` does not consult any servant locators or default servants.

See Also

- The Active Servant Map
- Object Identity
- Object Adapter States
- Server Implementation Techniques
- Servant Locators
- Default Servants

Previous Next