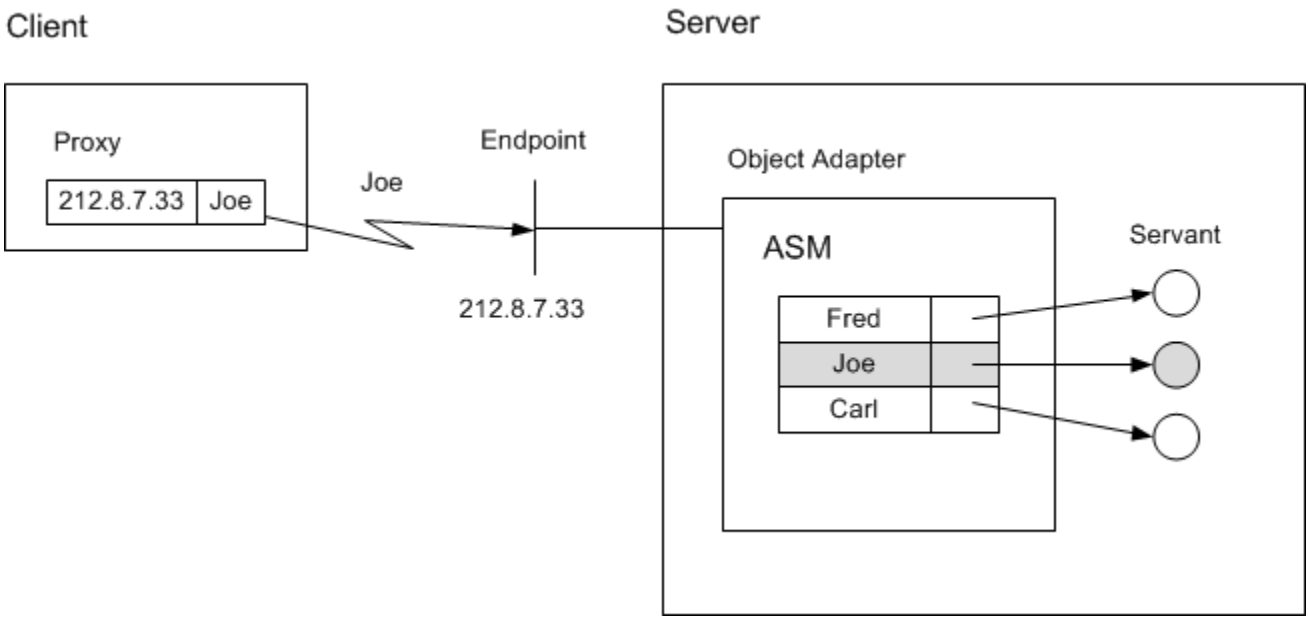
The Active Servant Map

Each object adapter maintains a data structure known as the active servant map.

On this page:

Role of the Active Servant Map

The *active servant map* (or *ASM*, for short) is a lookup table that maps object identities to servants: for C++, the lookup value is a smart pointer to the corresponding servant's location in memory; for Java and C#, the lookup value is a reference to the servant. When a client sends an operation invocation to the server, the request is targeted at a specific transport endpoint. Implicitly, the transport endpoint identifies the object adapter that is the target of the request (because no two object adapters can be bound to the same endpoint). The proxy via which the client sends its request contains the object identity for the corresponding object, and the client-side run time sends this object identity over the wire with the invocation. In turn, the object adapter uses that object identity to look in its ASM for the correct servant to dispatch the call to, as shown below:



Binding a request to the correct servant.

The process of associating a request via a proxy to the correct servant is known as *binding*. The scenario depicted in the illustration shows direct binding, in which the transport endpoint is embedded in the proxy. Ice also supports an indirect binding mode, in which the correct transport endpoints are provided by the IceGrid service.

If a client request contains an object identity for which there is no entry in the adapter's ASM, the adapter returns an `ObjectNotExistException` to the client (unless you use a default servant or servant locator).

Design Considerations for the Active Servant Map

Using an adapter's ASM to map Ice objects to servants has a number of design implications:

- Each Ice object is represented by a different servant.

  > It is possible to register a single servant with multiple identities. However, there is little point in doing so because a default servant achieves the same thing.

- All servants for all Ice objects are permanently in memory.

Using a separate servant for each Ice object in this fashion is common to many server implementations: the technique is simple to implement and provides a natural mapping from Ice objects to servants. Typically, on start-up, the server instantiates a separate servant for each Ice object, activates each servant, and then calls `activate` on the object adapter to start the flow of requests.

There is nothing wrong with the above design, provided that two criteria are met:

1. The server has sufficient memory available to keep a separate servant instantiated for each Ice object at all times.
2. The time required to initialize all the servants on start-up is acceptable.

For many servers, neither criterion presents a problem: provided that the number of servants is small enough and that the servants can be initialized quickly, this is a perfectly acceptable design. However, the design does not scale well: the memory requirements of the server grow linearly with the number of Ice objects so, if the number of objects gets too large (or if each servant stores too much state), the server runs out of memory.

Ice offers two APIs that help you scale servers to larger numbers of objects: *servant locators* and *default servants*. A default servant is essentially a simplified version of a servant locator that satisfies the majority of use cases, whereas a servant locator provides more flexibility for those applications that require it.

See Also

- Servant Locators
- Default Servants
- IceGrid