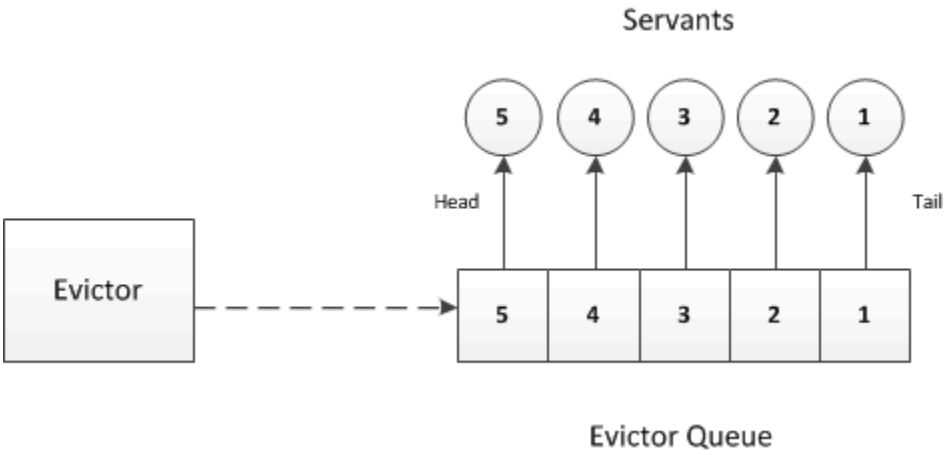
Servant Evictors

Previous Next

A particularly interesting use of a servant locator is as an *evictor* [1]. An evictor is a servant locator that maintains a cache of servants:

- Whenever a request arrives (that is, `locate` is called by the Ice run time), the evictor checks to see whether it can find a servant for the request in its cache. If so, it returns the servant that is already instantiated in the cache; otherwise, it instantiates a servant and adds it to the cache.
- The cache is a queue that is maintained in least-recently used (LRU) order: the least-recently used servant is at the tail of the queue, and the most-recently used servant is at the head of the queue. Whenever a servant is returned from or added to the cache, it is moved from its current queue position to the head of the queue, that is, the "newest" servant is always at the head, and the "oldest" servant is always at the tail.
- The queue has a configurable length that corresponds to how many servants will be held in the cache; if a request arrives for an Ice object that does not have a servant in memory and the cache is full, the evictor removes the least-recently used servant at the tail of the queue from the cache in order to make room for the servant about to be instantiated at the head of the queue.

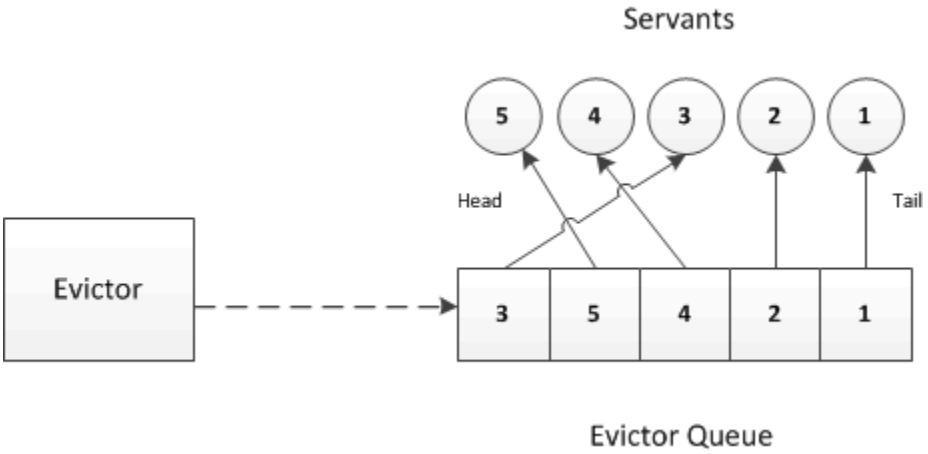
The figure below illustrates an evictor with a cache size of five after five invocations have been made, for object identities 1 to 5, in that order.



An evictor after five invocations for object identities 1 to 5.

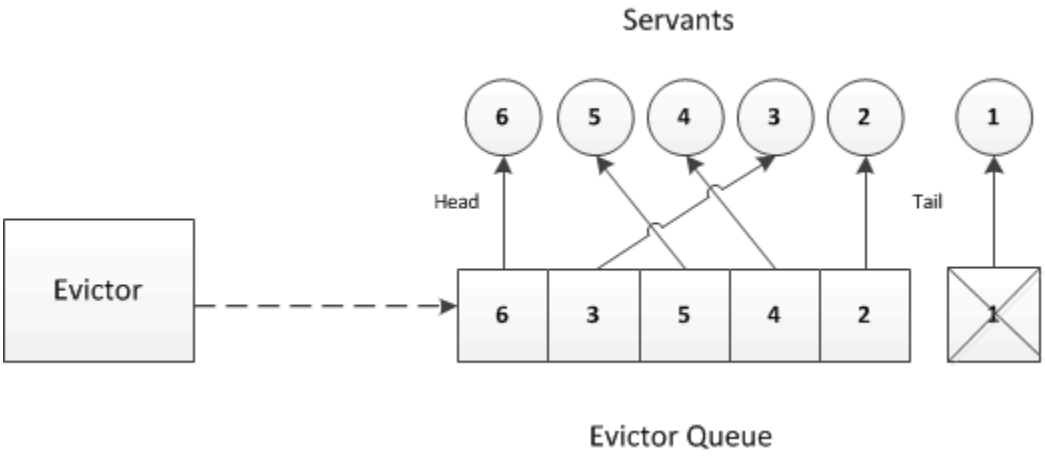
At this point, the evictor has instantiated five servants, and has placed each servant onto the evictor queue. Because requests were sent by the client for object identities 1 to 5 (in that order), servant 5 ends up at the head of the queue (at the most-recently used position), and servant 1 ends up at the tail of the queue (at the least-recently used position).

Assume that the client now sends a request for servant 3. In this case, the servant is found on the evictor queue and moved to the head position. The resulting ordering is shown below:



The evictor after accessing servant 3.

Assume that the next client request is for object identity 6. The evictor queue is fully populated, so the evictor creates a servant for object identity 6, places that servant at the head of the queue, and evicts the servant with identity 1 (the least-recently used servant) at the tail of the queue, as you can see here:



The evictor after evicting servant 1.

The evictor pattern combines the advantages of the ASM with the advantages of a default servant: provided that the cache size is sufficient to hold the working set of servants in memory, most requests are served by an already instantiated servant, without incurring the overhead of creating a servant and accessing the database to initialize servant state. By setting the cache size, you can control the trade-off between performance and memory consumption as appropriate for your application.

The following pages show how to implement an evictor in several languages. (You can also find the source code for the evictor with the code examples for this manual in the Ice distribution.)

Topics

Back to Top ^

See Also

- Servant Locators
- Default Servants

References

1. Henning, M., and S. Vinoski. 1999. Advanced CORBA Programming with C++. Reading, MA: Addison-Wesley.

Previous Next