Published on The O'Reilly Network (http://www.oreillynet.com/)
http://www.oreillynet.com/pub/a/onjava/2001/10/02/ejb.html
See this if you're having trouble printing code examples

JAVA COLUMN EJB 2 Tyler Jewell

Stateful Session EJBs: Beasts of Burden

10/02/2001

That's right. You heard it here first. I'm calling Stateful Session EJBs (SFSBs) the J2EE beasts of burden. I don't care for them. I despise them. I really hate them.

I don't dislike what they offer to developers. Rather, I dislike their conniving, misleading, and subtle nature that permits developers to make some of the worst architecture decisions I've seen. SFSBs have a very limited use in development and should rarely be seen. But somehow, these infiltrating creatures have wormed their way into designs where they do not belong. This article discusses the true purpose of SFSBs, where they belong in a J2EE design, and the situations where they should not be used. Heed this advice and perhaps we can harness the beast to create more robust architectures instead of creating more burdens by using SFSBs incorrectly.

The intent of Stateful Session EJBs

An Stateful Session EJB is a server-side object designed to hold data on behalf of a particular client. SFSBs should be used to store session-oriented data. Session-oriented data is used to track a multi-request sequence, ordering, or any associated data that is part of a sequence. SFSBs, however, have a limited lifespan and are *not* intended to survive server crashes. SFSBs are the only type of EJB that can receive callback notifications about the lifecycle of transactions that the bean participates in.

When SFSBs should not be used

- SFSBs are not server-side data caches. Yes, they do hold data in the server on behalf of the client, but that does not make them an in-memory, persistent store data cache. Yes, SFSBs are allowed to access a database and load data into memory, but the act of cacheing persistent data lies within the responsibilities of entity beans. A primary requirement for a data cache is that multiple clients should be able to access the same data concurrently based upon locking rules dictated by the developer. SFSBs are not designed to be shared resources to intuitively allow for this type of concurrent access. In the EJB 1.1 specification, concurrent client access of SFSBs was strictly prohibited. In EJB 2.0, a container vendor's implementation may permit concurrent client access of a single SFSB. Despite this new capability in EJB 2.0, SFSBs should not be used in a design that calls for multiple client concurrent access because they make it easy for a developer to code a deadlock.

  The only data that should be cached within an SFSB is data pertaining to tracking the sequence that the SFSB is managing. I've seen too many clients take 30 minutes to build an SFSB with persistent data and then spend 30 days trying to architect a solution that converts the bean into a transaction-aware data cache. Just don't do it -- you are setting yourself up for guaranteed failure.

  Part of this misperception about SFSBs comes from the wording within the EJB specification itself! The EJB specification suggests that a shopping cart for an e-commerce system could be implemented with SFSBs. This is perfectly acceptable if that shopping cart is only intended to be an in-memory implementation that does not need to survive a server crash. In reality, however, most implementations of shopping carts are required to survive server crashes; the data contained within the shopping cart needs to be persistent and transactional -- with an in-memory data cache.

Read Online -- Safari

Developers may be confused because if you read the EJB specification incorrectly it could be interpreted as saying all shopping cart implementations can be implemented with SFSBs. That simply isn't true. All in-memory shopping cart implementations can be implemented with SFSBs, but if you need to implement a persistent, crash-friendly shopping cart, you need to use entity EJBs.

- SFSBs are not distributed data stores. Yes, many application servers provide SFSB replication of state to allow for failover of requests to provide a higher level of availability. SFSB replication is just that: a replication mechanism to resist failures of instances on a single machine. SFSB replication is not designed to resist cluster-wide failures nor is it designed to cache data across an entire cluster. If a cluster has SFSB replication and all of the servers hosting a particular SFSB crash, that SFSB is permanently lost. This will happen, even if there are other servers that are still active. This is perfectly acceptable and should be accounted for when using SFSBs. If an SFSB is only used as a session-oriented component, the worst-case scenario that occurs is that the sequence is restarted from scratch when all copies of a particular SFSB are destroyed.

When SFSBs should be used in non-web systems

For systems that do not have a servlet/JSP front-end, SFSBs should be used to track session-oriented state similar to the way a web-based system would use an `HttpSession` object. This was the primary intent of SFSBs when they were created.

When SFSBs should be used in web systems

Systems that have JSP/servlet front-ends should use `HttpSession` objects to store session-oriented state on behalf of a client. Applications that manage an `HttpSession` object and an SFSB for a single client wind up duplicating effort that does not need to be duplicated. There are two reasons to use an SFSB in conjunction with an `HttpSession` object:

- Your application server does not provide cache management of `HttpSession` instances and your system is expected to have a large number of concurrent clients. Containers for SFSBs can activate and "passivate" the state of instances to and from a secondary store. This allows a container to create an upper limit to the number of instances that will exist in memory at any given point in time. The number of concurrent clients can exceed the limit of SFSB instances in memory because the container can swap idle instances to and from the secondary store. The container will never allow more than the limit of SFSB instances to exist in memory, subsequently placing all additional instances into the secondary store. This provides a greater level of scalability to the system through effective memory management.

  Many application servers provide similar cache management of `HttpSession` objects. Because `HttpSession` objects are similar to SFSBs, they can also be made passive and active. Cache management behavior of `HttpSession` objects is not required as part of J2EE and is considered a vendor value-add. *If your application server does not support `HttpSession` cache management -- and you need to control the total number of session-oriented instances in memory at any given time -- you should place the bulk of your session-oriented data in an SFSB instead of an `HttpSession` object.* You will still need to maintain an `HttpSession` for each client, but the only item in the `HttpSession` should be a reference to the SFSB for that client. If the only item in the `HttpSession` object is a reference to the SFSB, the amount of memory consumed by each `HttpSession` object is minimal and cache management of these instances is not needed. The bulk of memory consumption will occur within the SFSBs, which have a standardized strategy for allowing a container to perform cache management.

- Your session-oriented objects need to receive notifications on the lifecycle of the transactions they participate in. SFSBs can implement the `SessionSynchronization` interface. The `SessionSynchronization` interface contains three methods that a container invokes as a transaction migrates through the lifecycle the SFSB is participating in. An SFSB might implement the `SessionSynchronization` interface as a way to return the data of the SFSB to its original state whenever there is a transaction rollback. `HttpSession` instances do not have a mechanism that allows them to receive transaction notifications. *This means that any data that is modified in an `HttpSession` during a transaction will not be reverted if the current transaction is rolled back. All changes to data in an `HttpSession` object are always durable despite the outcome of any executing transactions.* If this behavior is not appropriate for your system, placing all data into an SFSB instance that implements `SessionSynchronization` will give you the appropriate behavior.

Conclusion

SFSBs have their place in J2EE systems; it's just very small. If your designs use SFSBs for session-oriented data and avoid using SFSBs as distributed data caches, your systems will be more reliable. If you have a requirement you think SFSBs can satisfy and that requirement is not listed in this article, you should give some hard thought to another solution.

Tyler Jewell , Director, Technical Evangelism, BEA Systems Tyler oversees BEA's technology evangelism efforts that are focused on driving early adoption of strategic BEA technologies into the ISV and developer community.

---

*Read more **EJB 2** columns.*

Return to ONJava.com.