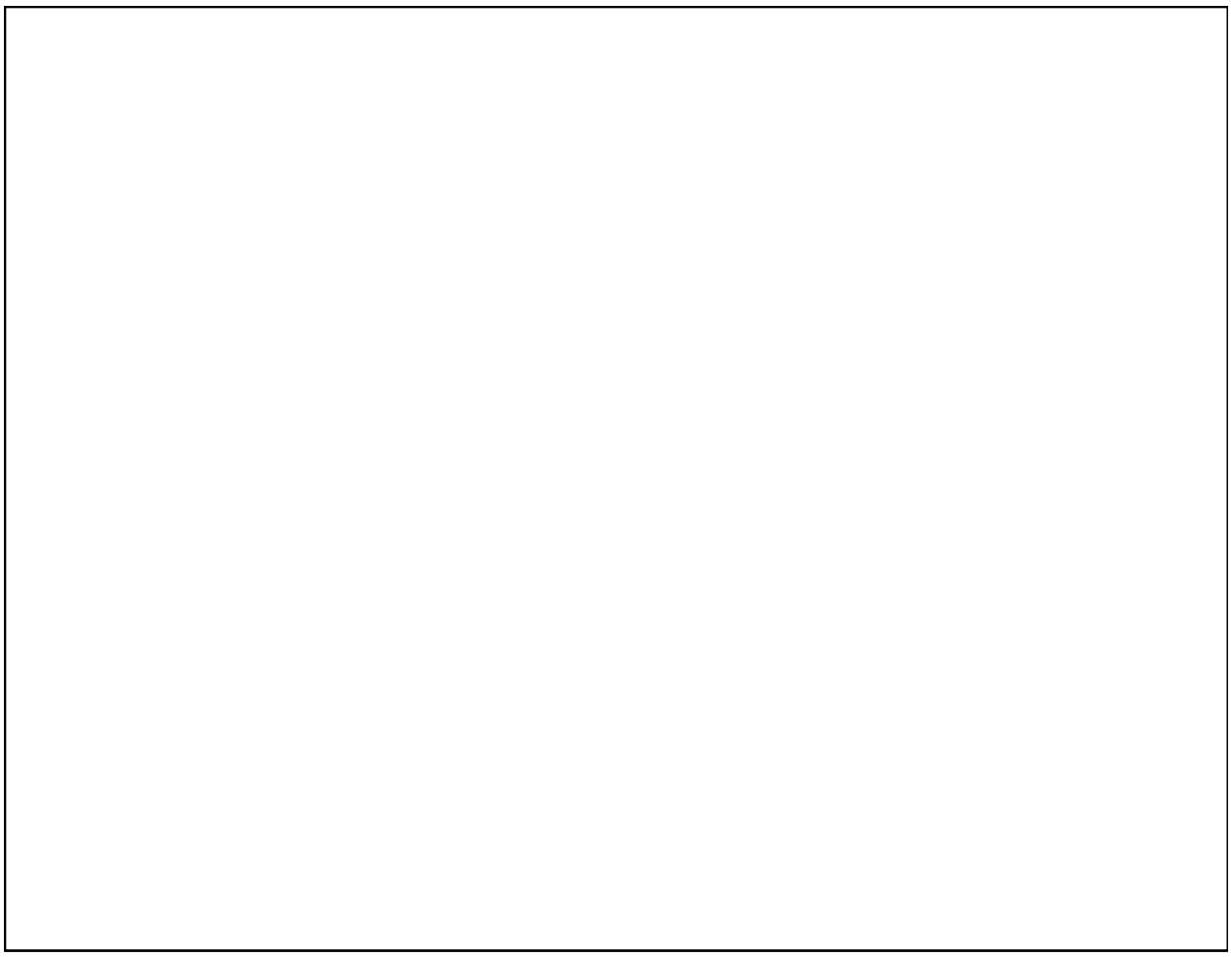
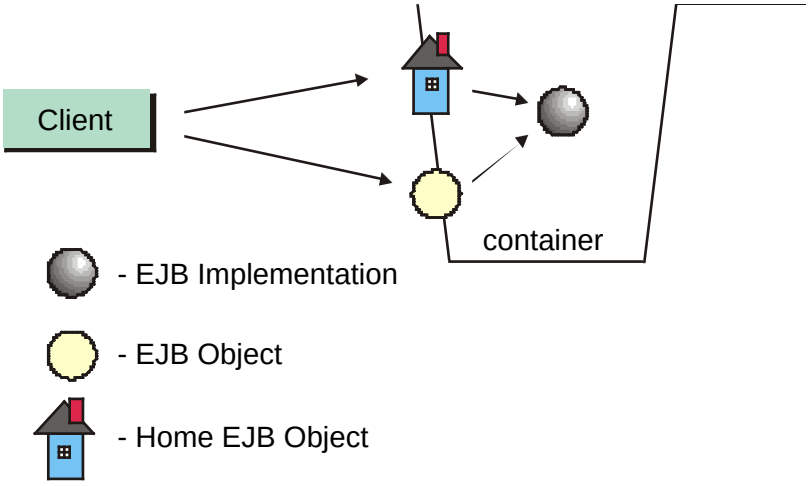
Stateless Session Beans

- Instance Pooling
- Stateless Session Bean Lifecycle



What is a Stateless Session Bean?

- Stateless session beans represent conversations that take exactly one method
- That one method can be arbitrarily complex



6 - 2

Remember that session beans represent business logic and processes -- stateless session beans are for the times when you can distill a process into a single method. Note that the method itself does not have to be simple -- it can perform tasks that take multiple operations, including accessing databases, other EJBs and so forth. The only issue is that the bean does not remember any state from one method to the next.

This may seem like a severe restriction, but it allows for some useful performance benefits for the container.

No Conversational State

- While stateless session beans can define instance variables, they must not modify that state on behalf of client
- Nor can clients depend on state staying the same between one method and the next
- Since they have no conversational state, each bean of a given type is identical and interchangeable
- Side effect: all necessary data to complete the method must be passed as method arguments

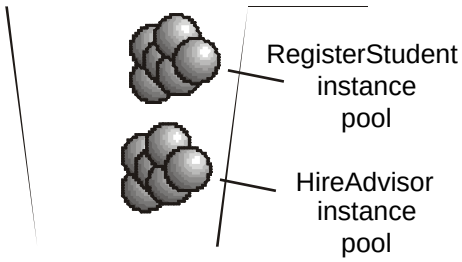
6 - 3

This is the essence of statelessness.

The bean can use instance variables to remember information, especially configuration information retrieved when the bean is initialized, but the bean must not modify the data on behalf of a client.

Stateless Session Bean Instance Pools

- Since stateless session beans are interchangeable, the container can pre-create a pool of EJB implementations
- The container must create a separate pool for each type of bean
- Containers can then efficiently assign EJB objects to a pool instance on a method-by-method basis

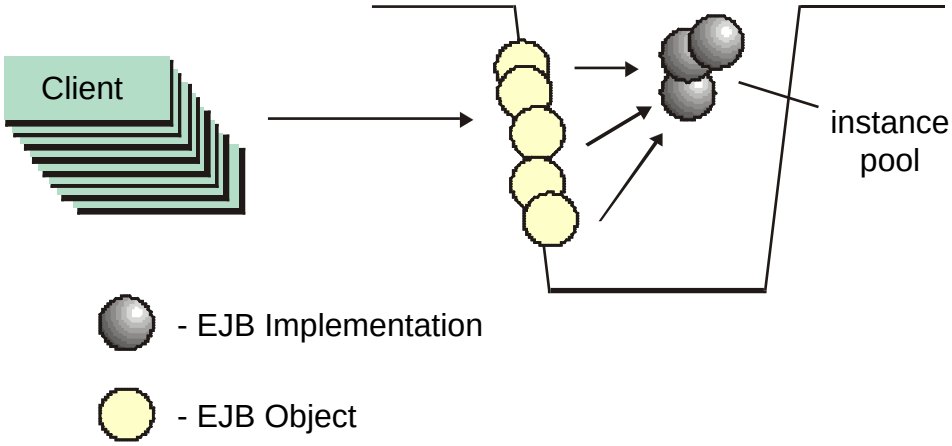


6 - 4

The EJB specification describes the methods and interactions of pooling, but does not detail how a container should implement an instance pool. That lets container vendors compete (and trade-off) on pool implementations. For example, some containers may dynamically change the pool size based on demand. Check your container's documentation for details.

The Benefit of Instance Pooling

- To save memory, containers can use a relatively small number of bean instances to service a relatively large number of clients



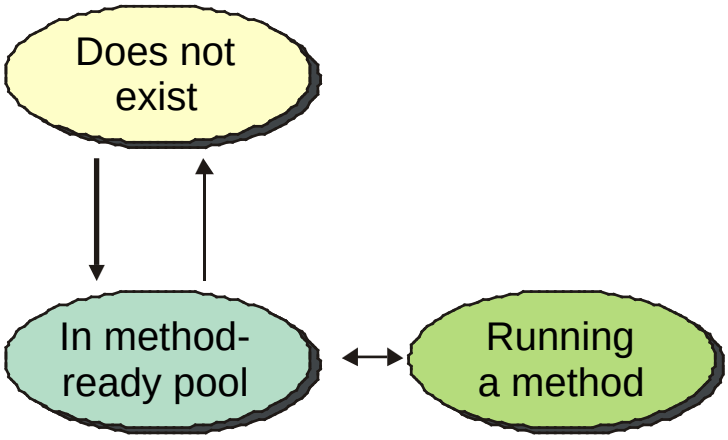
6 - 5

If your clients wait a while between methods and if methods execute relatively quickly, then pooling provides real benefits in memory usage. In other words, the small pool can be shared among large numbers of clients without clients needing to wait for an instance to become available.

However, if you have long-running methods, pooling is less efficient, and eventually collapses to using a single bean instance per client. If that's the case, then stateful beans may be a better choice, since they remember state between methods, letting you decompose a complex task into many simpler ones. There are penalties associated with stateful beans, however, that we will cover later.

Stateless Session Bean State Lifecycle

- The EJB Specification details the lifecycle of stateless session beans

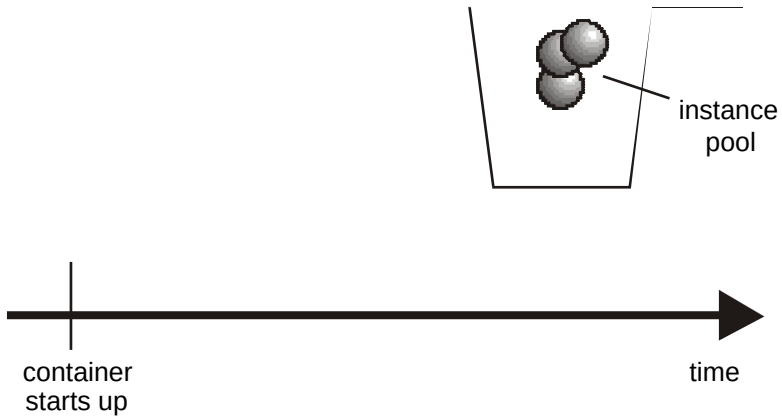


6 - 6

Stateless session beans have a well-defined lifecycle. The next few pages detail the state transitions.

Container Starts Up

- When the container starts up, it creates a pool of each stateless bean type
- The beans are in the *method-ready pooled* state



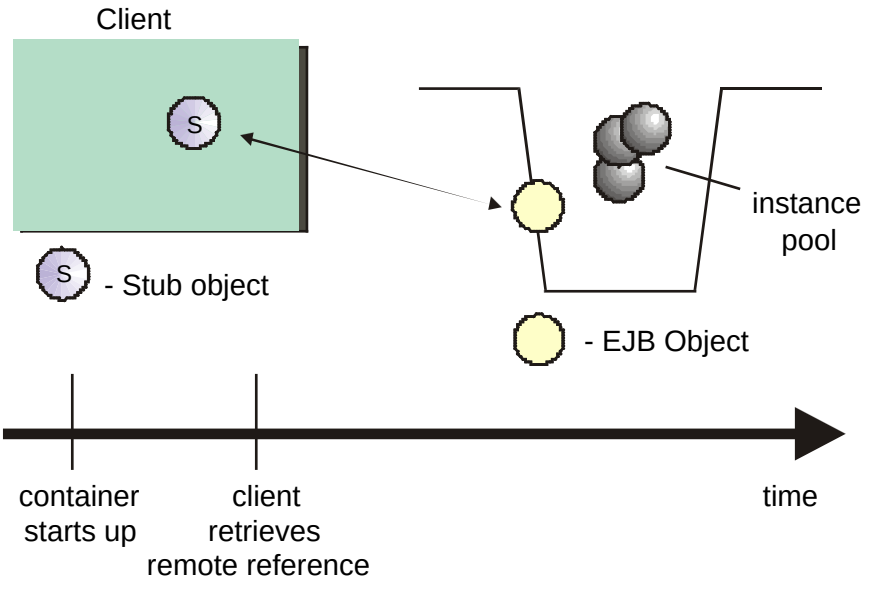
6 - 7

When the container creates each bean instance, it calls setSessionContext() and ejbCreate() on the new instance, giving the instance a chance to perform one-time initialization.

Note that the container can also add bean instances dynamically based on demand -- if so, it calls the same methods.

Client Retrieves Remote Reference

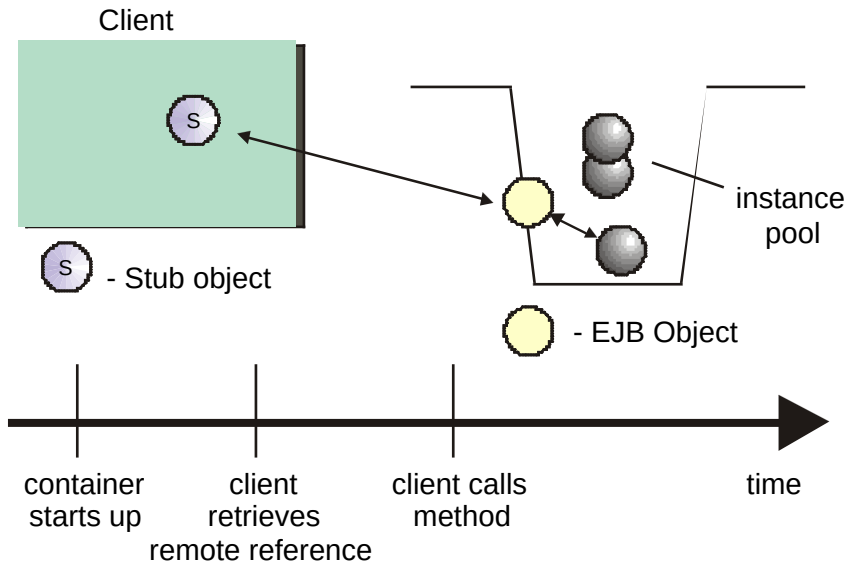
- When the client calls the home's create() method, the container creates an EJB object and returns its stub to the client



When the client uses the home to retrieve a reference, the container creates an EJB object, but doesn't necessarily assign the EJB object to a bean instance. In other words, no bean instance changes state at this point.

Client Calls a Remote Method

- When the client calls a method on the EJB object, the container assigns a bean instance to the EJB object and dispatches the method



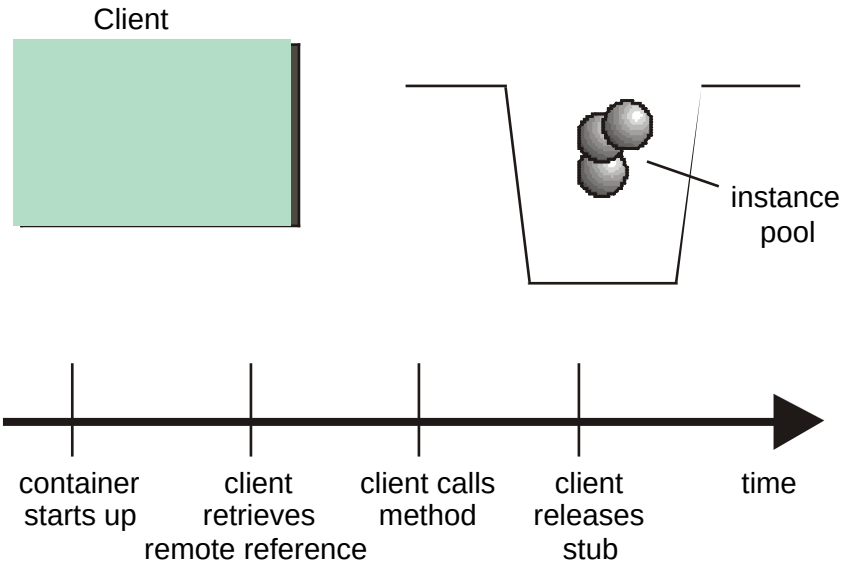
6 - 9

The transition from method-ready to running occurs when the client calls a method. The container choose a bean instance, runs the method, and then can return the instance to the pool. The algorithm that the container uses to choose an instance is container-specific.

When the method completes, the container can disassociate the bean instance from the EJB object -- the next method might use a different bean instance

Client Releases Remote Reference

- If the client explicitly removes the remote reference, the container deletes the EJB object



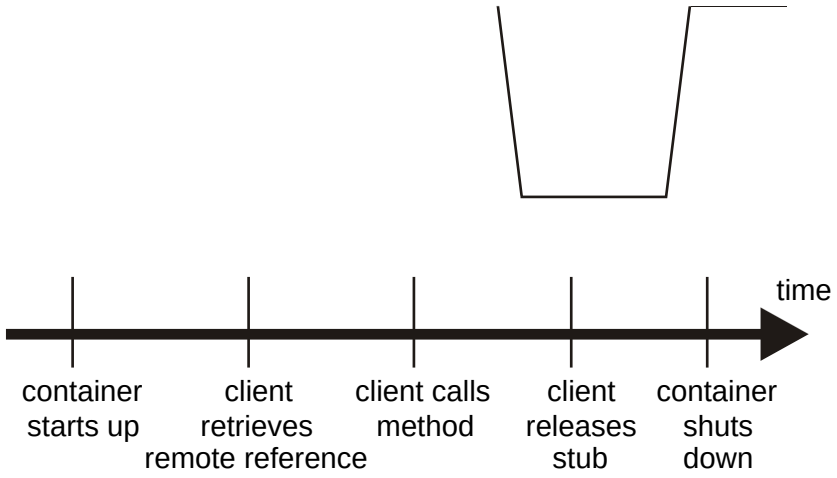
6 - 10

When the client is done with the remote reference, the container can delete the EJB object, but no state transition occurs on any bean instance.

To release the reference, clients can call the remove() method either on the bean reference or on the home that created the bean.

The Container Shuts Down

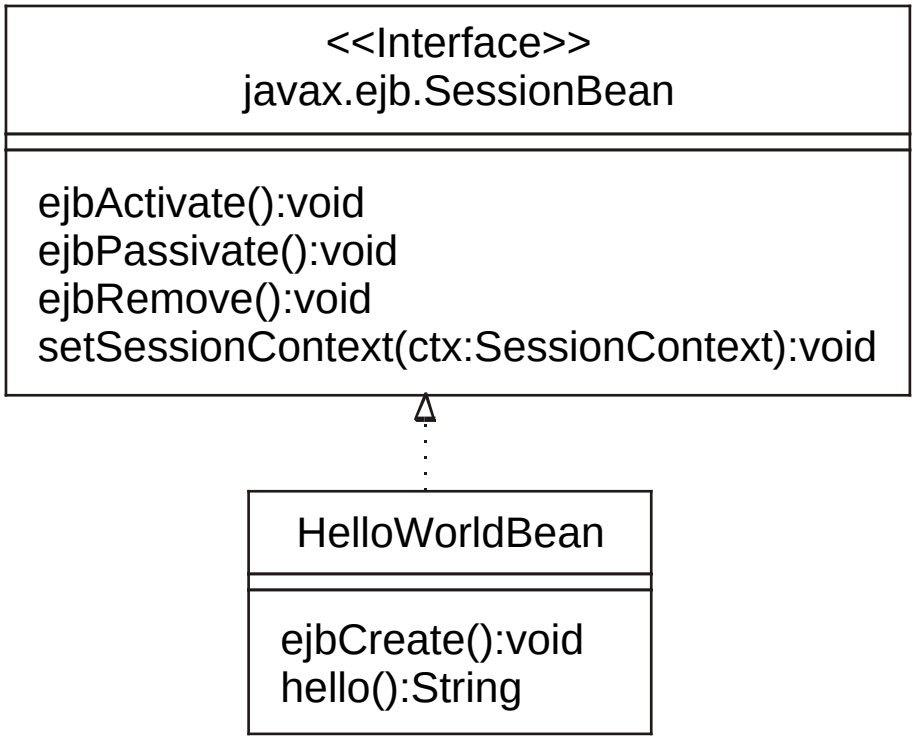
- When the container shuts down, it garbage-collects the bean instances
- The container can also remove instances from the pool dynamically based on demand



6 - 11

When the container shuts down, or decides to remove a bean instance for any other reason, it calls the ejbRemove() method on the bean instance.

Session Bean Interfaces



6 - 12

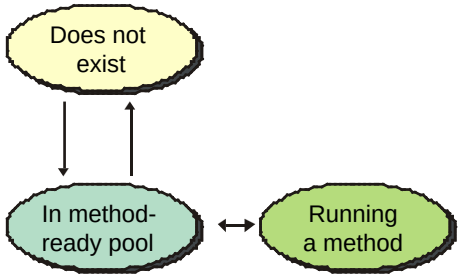
Remember that you must define a remote interface for an EJB and then write an implementation. This figure shows the inheritance hierarchy for the bean implementation. In addition to the framework methods shown here, the implementation must also provide methods for all of the methods defined in the remote interface (e.g. "hello()").

Note that stateless session beans must also provide an ejbCreate() method.

One more note: since stateless session beans have no conversational state, the container will never call the ejbActivate() and ejbPassivate() methods. Those methods are useful in stateful session beans. But even though they are never called, the stateless bean must provide at least an empty method body for each.

Stateless Session Bean Construction

- The home for a stateless session bean must define only the create() method -- the corresponding method in the bean implementation is ejbCreate()
- The container calls this method once in the instance's lifetime: when the instance is added to the pool
- You can use this method to perform one-time initialization tasks



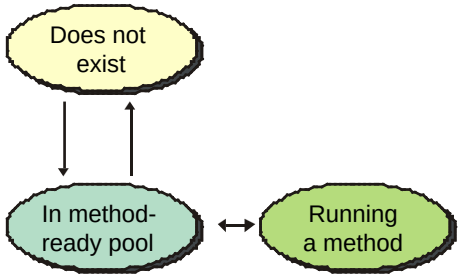
6 - 13

We will see later than other EJB types can have multiple creation and so-called "finder" methods, but stateless beans can only define the equivalent of a no-argument constructor. That makes sense if you remember that the container pre-creates the beans before clients ever have a chance to access them. In other words, by the time a client could pass arguments to a "constructor" method, the bean instance is already created. So the ejbCreate() method does not really work the same as a constructor in a local Java class.

But that doesn't mean that ejbCreate() is useless -- just that it must only perform initialization tasks that are not specific to any client. Examples of such tasks include connecting to a database or other legacy application, retrieving environment properties from the EJB context and so forth.

Stateless Session Bean Destruction

- Both the home and the remote interface define remove() methods
- When the client calls remove() on a stateless session bean, it affects the EJB object, not necessarily a bean instance
- The container calls ejbRemove() on the bean instance when it's deleted from the pool -- you can use this method to clean up resources allocated in ejbCreate()



Like ejbCreate(), the container does not call ejbRemove() in response to any action by the client. Instead, the container calls this method when it decides to garbage-collect the bean instance. The normal processing in ejbRemove() is to de-allocate or disconnect from any resources initialized during ejbCreate().

Chapter Summary

In this chapter, you learned:

- About stateless session bean instance pooling
- About the stateless bean lifecycle
- How to code the ejbCreate() and ejbRemove() lifecycle methods