Resources

All containers will be created automatically - which means you don't need to define them if you don't need to tune their configuration - when a bean of their type if found.

To avoid that use `openejb.offline` property and set it to `true`. See Server Configuration for more detail.

@Stateless

A `@Stateless` container.

Declarable in tomee.xml via

```
<Container id="Foo" type="STATELESS">
    AccessTimeout = 30 seconds
    MaxSize = 10
    MinSize = 0
    StrictPooling = true
    MaxAge = 0 hours
    ReplaceAged = true
    ReplaceFlushed = false
    MaxAgeOffset = -1
    IdleTimeout = 0 minutes
    GarbageCollection = false
    SweepInterval = 5 minutes
    CallbackThreads = 5
    CloseTimeout = 5 minutes
    UseOneSchedulerThreadByBean = false
    EvictionThreads = 1
</Container>
```

Declarable in properties via

```
Foo = new://Container?type=STATELESS
Foo.AccessTimeout = 30 seconds
Foo.MaxSize = 10
Foo.MinSize = 0
Foo.StrictPooling = true
Foo.MaxAge = 0 hours
Foo.ReplaceAged = true
Foo.ReplaceFlushed = false
Foo.MaxAgeOffset = -1
Foo.IdleTimeout = 0 minutes
Foo.GarbageCollection = false
Foo.SweepInterval = 5 minutes
Foo.CallbackThreads = 5
Foo.CloseTimeout = 5 minutes
Foo.UseOneSchedulerThreadByBean = false
Foo.EvictionThreads = 1
```

Configuration

AccessTimeout

Specifies the time an invokation should wait for an instance of the pool to become available.

After the timeout is reached, if an instance in the pool cannot be obtained, the method invocation will fail.

Usable time units: nanoseconds, microsecons, milliseconds, seconds, minutes, hours, days. Or any combination such as "1 hour and 27 minutes and 10 seconds"

Any usage of the `javax.ejb.AccessTimeout` annotation will override this setting for the bean or method where the annotation is used.

MaxSize

Specifies the size of the bean pools for this stateless SessionBean container. If StrictPooling is not used, instances will still be created beyond this number if there is demand, but they will not be returned to the pool and instead will be immediately destroyed.

MinSize

Specifies the minimum number of bean instances that should be in the pool for each bean. Pools are prefilled to the minimum on startup. Note this will create start order dependencies between other beans that also eagerly start, such as other `@Stateless` beans with a minimum or `@Singleton` beans using `@Startup`. The start order.

The minimum pool size is rigidly maintained. Instances in the minimum side of the pool are not eligible for `IdleTimeout` or `GarbageCollection`, but are subject to `MaxAge` and flushing.

If the pool is flushed it is immediately refilled to the minimum size with `MaxAgeOffset` applied. If an

instance from the minimum side of the pool reaches its `MaxAge`, it is also immediately replaced. Replacement is done in a background queue using the number of threads specified by `CallbackThreads`.

StrictPooling

StrictPooling tells the container what to do when the pool reaches it's maximum size and there are incoming requests that need instances.

With strict pooling, requests will have to wait for instances to become available. The pool size will never grow beyond the the set `MaxSize` value. The maximum amount of time a request should wait is specified via the `AccessTimeout` setting.

Without strict pooling, the container will create temporary instances to meet demand. The instances will last for just one method invocation and then are removed.

Setting `StrictPooling` to `false` and `MaxSize` to `0` will result in no pooling. Instead instances will be created on demand and live for exactly one method call before being removed.

MaxAge

Specifies the maximum time that an instance should live before it should be retired and removed from use. This will happen gracefully. Useful for situations where bean instances are designed to hold potentially expensive resources such as memory or file handles and need to be periodically cleared out.

Usable time units: nanoseconds, microsecons, milliseconds, seconds, minutes, hours, days. Or any combination such as `1 hour and 27 minutes and 10 seconds`

ReplaceAged

When `ReplaceAged` is enabled, any instances in the pool that expire due to reaching their `MaxAge` will be replaced immediately so that the pool will remain at its current size. Replacement is done in a background queue so that incoming threads will not have to wait for instance creation.

The aim of his option is to prevent user requests from paying the instance creation cost as `MaxAge` is enforced, potentially while under heavy load at peak hours.

Instances from the minimum side of the pool are always replaced when they reach their `MaxAge`, this setting dictates the treatment of non-minimum instances.

ReplaceFlushed

When `ReplaceFlushed` is enabled, any instances in the pool that are flushed will be replaced immediately so that the pool will remain at its current size. Replacement is done in a background queue so that incoming threads will not have to wait for instance creation.

The aim of his option is to prevent user requests from paying the instance creation cost if a flush performed while under heavy load at peak hours.

Instances from the minimum side of the pool are always replaced when they are flushed, this

setting dictates the treatment of non-minimum instances.

A bean may flush its pool by casting the `SessionContext` to `Flushable` and calling `flush()`. See `SweepInterval` for details on how flush is performed.

```
import javax.annotation.Resource;
import javax.ejb.SessionContext;
import javax.ejb.Stateless;
import java.io.Flushable;
import java.io.IOException;

public class MyBean {

    private SessionContext sessionContext;

    public void flush() throws IOException {

        ((Flushable) sessionContext).flush();
    }
}
```

MaxAgeOffset

Applies to MaxAge usage and would rarely be changed, but is a nice feature to understand.

When the container first starts and the pool is filled to the minimum size, all those "minimum" instances will have the same creation time and therefore all expire at the same time dictated by the `MaxAge` setting. To protect against this sudden drop scenario and provide a more gradual expiration from the start the container will spread out the age of the instances that fill the pool to the minimum using an offset.

The `MaxAgeOffset` is not the final value of the offset, but rather it is used in creating the offset and allows the spreading to push the initial ages into the future or into the past. The pool is filled at startup as follows:

```
for (int i = 0; i < poolMin; i++) {
    long ageOffset = (maxAge / poolMin * i * maxAgeOffset) % maxAge;
    pool.add(new Bean(), ageOffset));
}
```

The default `MaxAgeOffset` is -1 which causes the initial instances in the pool to live a bit longer before expiring. As a concrete example, let's say the MinSize is 4 and the MaxAge is 100 years. The generated offsets for the four instances created at startup would be 0, -25, -50, -75. So the first instance would be "born" at age 0, die at 100, living 100 years. The second instance would be born at -25, die at 100, living a total of 125 years. The third would live 150 years. The fourth 175 years.

A `MaxAgeOffset` of 1 would cause instances to be "born" older and therefore die sooner. Using the same example `MinSize` of 4 and `MaxAge` of `100 years`, the life spans of these initial four instances

would be 100, 75, 50, and 25 years respectively.

A `MaxAgeOffset` of 0 will cause no "spreading" of the age of the first instances used to fill the pool to the minimum and these instances will of course reach their MaxAge at the same time. It is possible to set to decimal values such as -0.5, 0.5, -1.2, or 1.2.

IdleTimeout

Specifies the maximum time that an instance should be allowed to sit idly in the pool without use before it should be retired and removed.

Usable time units: nanoseconds, microsecons, milliseconds, seconds, minutes, hours, days. Or any combination such as "1 hour and 27 minutes and 10 seconds"

GarbageCollection

Allows Garbage Collection to be used as a mechanism for shrinking the pool. When set to true all instances in the pool, excluding the minimum, are eligible for garbage collection by the virtual machine as per the rules of `java.lang.ref.SoftReference` and can be claimed by the JVM to free memory. Instances garbage collected will have their `@PreDestroy` methods called during finalization.

In the OpenJDK VM the `-XX:SoftRefLRUPolicyMSPerMB` flag can adjust how aggressively SoftReferences are collected. The default OpenJDK setting is 1000, resulting in inactive pooled instances living one second of lifetime per free megabyte in the heap, which is very aggressive. The setting should be increased to get the most out of the `GarbageCollection` feature of the pool. Much higher settings are safe. Even a setting as high as 3600000 (1 hour per free MB in the heap) does not affect the ability for the VM to garbage collect SoftReferences in the event that memory is needed to avoid an `OutOfMemoryException`.

SweepInterval

The frequency in which the container will sweep the pool and evict expired instances. Eviction is how the `IdleTimeout`, `MaxAge`, and pool "flush" functionality is enforced. Higher intervals are better.

Instances in use are excluded from sweeping. Should an instance expire while in use it will be evicted immediately upon return to the pool. Effectively `MaxAge` and flushes will be enforced as a part of normal activity or sweeping, while IdleTimeout is only enforcable via sweeping. This makes aggressive sweeping less important for a pool under moderate load.

Usable time units: nanoseconds, microsecons, milliseconds, seconds, minutes, hours, days. Or any combination such as `1 hour and 27 minutes and 10 seconds`

CallbackThreads

When sweeping the pool for expired instances a thread pool is used to process calling `@PreDestroy` on expired instances as well as creating new instances as might be required to fill the pool to the minimum after a Flush or `MaxAge` expiration. The `CallbackThreads` setting dictates the size of the thread pool and is shared by all beans deployed in the container.

CloseTimeout

PostConstruct methods are invoked on all instances in the pool when the bean is undeployed and its pool is closed. The `CloseTimeout` specifies the maximum time to wait for the pool to close and `PostConstruct` methods to be invoked.

Usable time units: nanoseconds, microsecons, milliseconds, seconds, minutes, hours, days. Or any combination such as `1 hour and 27 minutes and 10 seconds`

UseOneSchedulerThreadByBean

back to previous behavior (TomEE 1.x) where 1 scheduler thread was used for stateless eviction by bean (ie for 500 stateless beans you get 500 eviction threads)

EvictionThreads

number of threads to associate to eviction threads (1 is not bad for most applications)

@Stateful

A `@Stateful` container.

Declarable in tomee.xml via

```
<Container id="Foo" type="STATEFUL">
    AccessTimeout = 30 seconds
    Cache = org.apache.openejb.core.stateful.SimpleCache
    Passivator = org.apache.openejb.core.stateful.SimplePassivater
    TimeOut = 20
    Frequency = 60
    Capacity = 1000
    BulkPassivate = 100
</Container>
```

Declarable in properties via

```
Foo = new://Container?type=STATEFUL
Foo.AccessTimeout = 30 seconds
Foo.Cache = org.apache.openejb.core.stateful.SimpleCache
Foo.Passivator = org.apache.openejb.core.stateful.SimplePassivater
Foo.TimeOut = 20
Foo.Frequency = 60
Foo.Capacity = 1000
Foo.BulkPassivate = 100
```

Configuration

AccessTimeout

Specifies the maximum time an invocation could wait for the `@Stateful` bean instance to become available before giving up.

After the timeout is reached a `javax.ejb.ConcurrentAccessTimeoutException` will be thrown.

Usable time units: nanoseconds, microsecons, milliseconds, seconds, minutes, hours, days. Or any combination such as "1 hour and 27 minutes and 10 seconds"

Any usage of the `javax.ejb.AccessTimeout` annotation will override this setting for the bean or method where the annotation is used.

Cache

The cache is responsible for managing stateful bean instances. The cache can page instances to disk as memory is filled and can destroy abandoned instances. A different cache implementation can be used by setting this property to the fully qualified class name of the Cache implementation.

Passivator

The passivator is responsible for writing beans to disk at passivation time. Different passivators can be used by setting this property to the fully qualified class name of the `PassivationStrategy` implementation. The passivator is not responsible for invoking any callbacks or other processing, its only responsibly is to write the bean state to disk.

Known implementations:

- org.apache.openejb.core.stateful.RAFPassivater
- org.apache.openejb.core.stateful.SimplePassivater

TimeOut

Specifies the time a bean can be idle before it is removed by the container.

This value is measured in minutes. A value of 5 would result in a time-out of 5 minutes between invocations. A value of -1 would mean no timeout. A value of 0 would mean a bean can be immediately removed by the container.

Any usage of the `javax.ejb.StatefulTimeout` annotation will override this setting for the bean where the annotation is used.

Frequency

Specifies the frequency (in seconds) at which the bean cache is checked for idle beans.

Capacity

Specifies the size of the bean pools for this stateful SessionBean container.

BulkPassivate

Property name that specifies the number of instances to passivate at one time when doing bulk passivation.

@Singleton

A `@Singleton` container.

Declarable in tomee.xml via

```
<Container id="Foo" type="SINGLETON">
    AccessTimeout = 30 seconds
</Container>
```

Declarable in properties via

```
Foo = new://Container?type=SINGLETON
Foo.AccessTimeout = 30 seconds
```

Configuration

AccessTimeout

Specifies the maximum time an invocation could wait for the `@Singleton` bean instance to become available before giving up.

After the timeout is reached a `javax.ejb.ConcurrentAccessTimeoutException` will be thrown.

Usable time units: nanoseconds, microsecons, milliseconds, seconds, minutes, hours, days. Or any combination such as `1 hour and 27 minutes and 10 seconds`

Any usage of the `javax.ejb.AccessTimeout` annotation will override this setting for the bean or method where the annotation is used.

@MessageDriven

A MDB container.

Declarable in tomee.xml via

```
<Container id="Foo" type="MESSAGE">
    ResourceAdapter = Default JMS Resource Adapter
    MessageListenerInterface = javax.jms.MessageListener
    ActivationSpecClass = org.apache.activemq.ra.ActiveMQActivationSpec
    InstanceLimit = 10
    FailOnUnknowActivationSpec = true
</Container>
```

Declarable in properties via

```
Foo = new://Container?type=MESSAGE
Foo.ResourceAdapter = Default JMS Resource Adapter
Foo.MessageListenerInterface = javax.jms.MessageListener
Foo.ActivationSpecClass = org.apache.activemq.ra.ActiveMQActivationSpec
Foo.InstanceLimit = 10
Foo.FailOnUnknowActivationSpec = true
```

Configuration

ResourceAdapter

The resource adapter delivers messages to the container

MessageListenerInterface

Specifies the message listener interface handled by this container

ActivationSpecClass

Specifies the activation spec class

InstanceLimit

Specifies the maximum number of bean instances that are allowed to exist for each MDB deployment.

FailOnUnknowActivationSpec

Log a warning if true or throw an exception if false is an activation spec can't be respected

@Managed

A managed bean container.

Declarable in tomee.xml via

```
<Container id="Foo" type="MANAGED" />
```

Declarable in properties via

```
Foo = new://Container?type=MANAGED
```

CMP entity

A CMP bean container.

Declarable in tomee.xml via

```
<Container id="Foo" type="CMP_ENTITY">
    CmpEngineFactory = org.apache.openejb.core.cmp.jpa.JpaCmpEngineFactory
</Container>
```

Declarable in properties via

```
Foo = new://Container?type=CMP_ENTITY
Foo.CmpEngineFactory = org.apache.openejb.core.cmp.jpa.JpaCmpEngineFactory
```

Configuration

CmpEngineFactory

The engine to use for this container. By default TomEE only provides the JPA implementation.

BMP entity

A BMP entity container.

Declarable in tomee.xml via

```
<Container id="Foo" type="BMP_ENTITY">
    PoolSize = 10
</Container>
```

Declarable in properties via

```
Foo = new://Container?type=BMP_ENTITY
Foo.PoolSize = 10
```

Configuration

PoolSize

Specifies the size of the bean pools for this bmp entity container.