@AccessTimeout annotation

Example access-timeout can be browsed at https://github.com/apache/tomee/tree/master/examples/access-timeout

Before taking a look at `@AccessTimeout`, it might help to see when a caller might have to "wait"

Waiting

Stateful Bean

NOTE	By default, clients are allowed to make concurrent calls to a stateful session object and the container is required to serialize such concurrent requests. Note that the container never permits multi-threaded access to the actual stateful session bean instance. For this reason, Read/Write method locking metadata, as well as the bean-managed concurrency mode, are not applicable to stateful session beans and must not be used.

This means that when a method `foo()` of a stateful bean instance is being executed and a second request comes in for that method or another method, EJB container would serialize the second request. It would not let the method be executed concurrently but wait until the first request is processed.

The client would wait also when `@Stateful` bean is in a transaction and the client is invoking it from outside that transaction.

Stateless Bean

Say there are 20 instances of a bean in the pool and all of them are busy. Now when the next request comes in, the processing **might wait** for a bean to be available in the pool. (Note: Pooling sematics, if any, are not covered by the spec. The vendor's pooling semantics might or might not involve a wait condition)

Singleton Bean

The container enforces a single-threaded access by default to singleton beans. That's the equivalent of annotating with `@Lock(Write)`. So when a `@Lock(Write)` method is executing, all other `@Lock(READ)` and `@Lock(WRITE)` method invocations would have to wait.

Summary

- `@Singleton` - an `@Lock(WRITE)` method is being invoked and container-managed concurrency is being used. All methods are `@Lock(WRITE)` by default.
- `@Stateful` - any method of the instance is being invoked and a second invocation occurs. OR the `@Stateful` bean is in a transaction and the caller is invoking it from outside that transaction.

- `@Stateless` - no instances are available in the pool. As noted, however, pooling sematics, if any, are not covered by the spec. If the vendor's pooling semantics do involve a wait condition, the @AccessTimeout should apply.

@AccessTimeout

The `@AccessTimeout` is simply a convenience wrapper around the `long` and `TimeUnit` tuples commonly used in the `java.util.concurrent` API.

```
import java.util.concurrent.TimeUnit;
@Target({METHOD, TYPE})
@Retention(RUNTIME)
public @interface AccessTimeout {
    long value();
    TimeUnit unit() default TimeUnit.MILLISECONDS;
}
```

Usage

A method or class can be annotated with @AccessTimeout to specify the maximum time a call might wait for access to the bean wait should a wait condition occur.

The semantics of the value element are as follows:

- A `value` > 0 indicates a timeout value in the units specified by the `unit` element.
- A `value` of 0 means concurrent access is not permitted.
- A `value` of -1 indicates that the client request will block indefinitely until forward progress it can proceed.

Just as simple as that !

What exception would the client receive, with a timeout ?

Quoting from the spec, "if a client-invoked business method is in progress on an instance when another client-invoked call, from the same or different client, arrives at the same instance of a stateful session bean, if the second client is a client of the bean s business interface or no-interface view, the concurrent invocation must result in the second client receiving a javax.ejb.ConcurrentAccessException[15]. If the EJB 2.1 client view is used, the container must throw a java.rmi.RemoteException if the second client is a remote client, or a javax.ejb.EJBException if the second client is a local client"

No standard default

Note that the `value` attribute has no default. This was intentional and intended to communicate that if `@AccessTimeout` is not explicitly used, the behavior you get is vendor-specific.

Some vendors will wait for a preconfigured time and throw `javax.ejb.ConcurrentAccessException`, some vendors will throw it immediately.

Example

Here we have a simple @Singleton bean that has three synchronous methods and one `@Asynchronous` method. The bean itself is annotated `@Lock(WRITE)` so that only one thread may access the `@Singleton` at a time. This is the default behavior of an `@Singleton` bean, so explicit usage of `@Lock(WRITE)` is not needed but is rather nice for clarity as the single-threaded nature of the bean is important to the example.

```
@Singleton
@Lock(WRITE)
public class BusyBee {

    @Asynchronous
    public Future stayBusy(CountDownLatch ready) {
        ready.countDown();

        try {
            new CountDownLatch(1).await();
        } catch (InterruptedException e) {
            Thread.interrupted();
        }

        return null;
    }

    @AccessTimeout(0)
    public void doItNow() {
        // do something
    }

    @AccessTimeout(value = 5, unit = TimeUnit.SECONDS)
    public void doItSoon() {
        // do something
    }

    @AccessTimeout(-1)
    public void justDoIt() {
        // do something
    }

}
```

The `@Asynchronous` method is not a critical part of `@AccessTimeout`, but serves as a simple way to "lock" the bean for testing purposes. It allows us to easily test the concurrent behavior of the bean.

```
public class BusyBeeTest extends TestCase {

    public void test() throws Exception {

        final Context context = EJBContainer.createEJBContainer().getContext();

        final CountDownLatch ready = new CountDownLatch(1);

        final BusyBee busyBee = (BusyBee) context.lookup("java:global/access-
timeout/BusyBee");

        // This asynchronous method will never exit
```

```
        busyBee.stayBusy(ready);

        // Are you working yet little bee?
        ready.await();


        // OK, Bee is busy


        { // Timeout Immediately
            final long start = System.nanoTime();

            try {
                busyBee.doItNow();

                fail("The bee should be busy");
            } catch (Exception e) {
                // the bee is still too busy as expected
            }

            assertEquals(0, seconds(start));
        }

        { // Timeout in 5 seconds
            final long start = System.nanoTime();

            try {
                busyBee.doItSoon();

                fail("The bee should be busy");
            } catch (Exception e) {
                // the bee is still too busy as expected
            }

            assertEquals(5, seconds(start));
        }

        // This will wait forever, give it a try if you have that long
        //busyBee.justDoIt();
    }

    private long seconds(long start) {
        return TimeUnit.NANOSECONDS.toSeconds(System.nanoTime() - start);
    }
}
```

Running

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running org.superbiz.accesstimeout.BusyBeeTest
Apache OpenEJB 4.0.0-beta-1    build: 20111002-04:06
http://tomee.apache.org/
INFO - openejb.home = /Users/dblevins/examples/access-timeout
INFO - openejb.base = /Users/dblevins/examples/access-timeout
INFO - Using 'javax.ejb.embeddable.EJBContainer=true'
INFO - Configuring Service(id=Default Security Service, type=SecurityService,
provider-id=Default Security Service)
INFO - Configuring Service(id=Default Transaction Manager, type=TransactionManager,
provider-id=Default Transaction Manager)
INFO - Found EjbModule in classpath: /Users/dblevins/examples/access-
timeout/target/classes
INFO - Beginning load: /Users/dblevins/examples/access-timeout/target/classes
INFO - Configuring enterprise application: /Users/dblevins/examples/access-timeout
INFO - Configuring Service(id=Default Singleton Container, type=Container, provider-
id=Default Singleton Container)
INFO - Auto-creating a container for bean BusyBee: Container(type=SINGLETON,
id=Default Singleton Container)
INFO - Configuring Service(id=Default Managed Container, type=Container, provider-
id=Default Managed Container)
INFO - Auto-creating a container for bean org.superbiz.accesstimeout.BusyBeeTest:
Container(type=MANAGED, id=Default Managed Container)
INFO - Enterprise application "/Users/dblevins/examples/access-timeout" loaded.
INFO - Assembling app: /Users/dblevins/examples/access-timeout
INFO - Jndi(name="java:global/access-
timeout/BusyBee!org.superbiz.accesstimeout.BusyBee")
INFO - Jndi(name="java:global/access-timeout/BusyBee")
INFO -
Jndi(name="java:global/EjbModule748454644/org.superbiz.accesstimeout.BusyBeeTest!org.s
uperbiz.accesstimeout.BusyBeeTest")
INFO -
Jndi(name="java:global/EjbModule748454644/org.superbiz.accesstimeout.BusyBeeTest")
INFO - Created Ejb(deployment-id=org.superbiz.accesstimeout.BusyBeeTest, ejb-
name=org.superbiz.accesstimeout.BusyBeeTest, container=Default Managed Container)
INFO - Created Ejb(deployment-id=BusyBee, ejb-name=BusyBee, container=Default
Singleton Container)
INFO - Started Ejb(deployment-id=org.superbiz.accesstimeout.BusyBeeTest, ejb-
name=org.superbiz.accesstimeout.BusyBeeTest, container=Default Managed Container)
INFO - Started Ejb(deployment-id=BusyBee, ejb-name=BusyBee, container=Default
Singleton Container)
INFO - Deployed Application(path=/Users/dblevins/examples/access-timeout)
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 6.071 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```