



David Blevins | Apache

# Tomcat to JavaEE with Apache TomEE

@dblevins

[dblevins@apache.org](mailto:dblevins@apache.org)

@ApachTomEE / #TomEE

<http://tomee.apache.org>

What is Apache TomEE?

# Tomcat + JavaEE = TomEE

- Java EE 6 Web Profile certified
- Tomcat through and through
- All Apache components
  - OpenJPA
  - OpenWebBeans
  - OpenEJB
  - MyFaces
- Core Values
  - Be small
  - Be certified
  - Be Tomcat

# Flavors of TomEE

- Apache TomEE Web Profile (Java EE 6 Certified)
  - CDI
  - EJB
  - JPA
  - JSF
  - Bean Validation
- Apache TomEE Plus
  - JAX-RS
  - JAX-WS
  - JMS
- Embedded Apache TomEE

# Basic Stats

- Size
  - 27mb
- Memory usage
  - very little required
  - passes TCK with default 64mb
- Agility:
  - Eclipse deploy ~700ms
  - embedded test 2-4 seconds

# Certification

- Certified on Amazon EC2
  - t1.micro linux images,
  - 100 spot instances
  - 613mb memory
  - 64mb used, 549mb free
  - t1.micros have slow disks, memory is critical
- Current certified OSs
  - Amazon Linux AMI 2011.09, EBS boot, EC2 t1.micro
  - Amazon Linux AMI 2011.09, EBS boot, EC2 m1.small
  - Amazon Linux AMI 2011.09, EBS boot, EC2 c1.medium
- Runs daily
- Got a Cloud?
  - Donate time!

*“There is also one point to note. TomEE is insanely fast!”*

-- Łukasz Budnik, CXF JAX-RS on Apache TomEE @ DZone, May 23rd 2012

*“From an architect that switched from <XXX> to Tomee. I can tell you Tomee kicks <XXX>'s ass in every way. Memory, speed, reliability, validation, you name it.”*

-- Zeeman, User List, July 3rd 2012

*“If you are concerned about performance and footprint, but can accept that you'll have to solve problems yourself, go TomEE. (TomEE seems to be much faster than <XXX>)”*

-- Jonathan Fisher, User List, July 3rd 2012



# Gaps in Tomcat

# Gaps in Tomcat

- No Transaction support
- No Connection Pooling support
  - Pooling should be transactional
- No Integrated Security
- No support for Global JNDI
  - `java:module`
  - `java:app`
  - `java:global`
- No support for `@DataSourceDefinition`
- No support for new `<env-entry>` types:
  - `java.lang.Class`
  - Enums

# Gaps in Tomcat

- No @Resource
  - UserTransaction
  - BeanManager
  - Validator
  - ValidatorFactory
  - Topic/Queue
  - ConnectionFactory
- No @PersistenceUnit
- No @PersistenceContext
- No @Inject
- No @EJB
- No @WebServiceRef

# Migration Issues

# Building your own app server

- Including API jars in the webapp
  - JPA API
  - JSF API
  - etc.
- Including implementations in webapp
  - Mojarra (works in trunk)
  - Bitronix/Atomikos
- Non-compliant DataSource and JPA
  - too many years of Do It Yourself
  - providers not enforcing rules

# (Mis)Information

- Gaps in Knowledge
  - “that other stuff”
- Misinformation
  - don’t use X, it’s heavy
- Leads to...
  - let’s build own own heavy X!
- Bites you
  - non-portable apps
  - lacks tens of thousands of tests
  - lacks shared experience

# Closing Gaps in Knowledge

# You can't use this

```
public class MySeeminglySimpleBean {  
    @PersistenceContext  
    private EntityManager entityManager;  
    //... do some other things  
}
```

- unless you understand
  - JTA Transactions
  - Container-managed EntityManagers
  - Container-managed DataSources
  - How to get references to these, properly!



# Not Just EJBs

- Components with Transaction support
  - Servlets
  - JSP
  - JSF Managed Beans
  - CDI
  - EJB
  - JAX-RS REST Services
  - JAX-WS Web Services

# What are Transactions?

“Undo”

```
import javax.annotation.Resource;
import javax.transaction.UserTransaction;

public class TransactionBean {

    @Resource
    private UserTransaction userTransaction;

    public void doSomething() throws Exception {

        userTransaction.begin();
        try {

            // work, work, work

            // all I ever do is work

            userTransaction.commit();
        } catch (Throwable t) {

            // Undo!
            userTransaction.rollback();
        }
    }
}
```

```
@Resource
private UserTransaction userTransaction;

@PersistenceContext
private EntityManager entityManager;

@Resource
private DataSource dataSource;

public void testCommit() throws Exception {

    userTransaction.begin();
    try {
        // Use the EntityManager
        entityManager.persist(new Movie("Quentin Tarantino", "Reservoir Dogs", 1992));
        entityManager.persist(new Movie("Joel Coen", "Fargo", 1996));
        entityManager.persist(new Movie("Joel Coen", "The Big Lebowski", 1998));

        // Use the DataSource
        final Connection connection = dataSource.getConnection();
        final Statement statement = connection.createStatement();
        statement.execute(
            "INSERT INTO Movie (director, title, year, id) " +
            "VALUES ('Ethan Coen', 'True Grit', 2010, 4)");
    } finally {
        userTransaction.commit();
    }

    // Transaction was committed
    List<Movie> list = movies.getMovies();
    assertEquals("List.size()", 4, list.size());
}
```

```
@Resource
private UserTransaction userTransaction;

@PersistenceContext
private EntityManager entityManager;

@Resource
private DataSource dataSource;

public void testRollback() throws Exception {

    userTransaction.begin();
    try {
        // Use the EntityManager
        entityManager.persist(new Movie("Quentin Tarantino", "Reservoir Dogs", 1992));
        entityManager.persist(new Movie("Joel Coen", "Fargo", 1996));
        entityManager.persist(new Movie("Joel Coen", "The Big Lebowski", 1998));

        // Use the DataSource
        final Connection connection = dataSource.getConnection();
        final Statement statement = connection.createStatement();
        statement.execute(
            "INSERT INTO Movie (director, title, year, id) " +
            "VALUES ('Ethan Coen', 'True Grit', 2010, 4)");
    } finally {
        userTransaction.rollback(); // undo!
    }

    // Transaction was committed
    List<Movie> list = movies.getMovies();
    assertEquals("List.size()", 0, list.size()); // No movies
}
```

# Controlling Transactions

- `javax.transaction.UserTransaction`
  - Manual approach
  - Any JavaEE component
- `@javax.ejb.TransactionAttribute`
  - Class or method annotation
  - EJB only
- `@javax.transaction.Transactional`
  - Class or method annotation
  - Any JavaEE component\*
  - Coming in JavaEE 7

# Transaction Hooks

- SessionSynchronization
  - @AfterBegin
  - @BeforeCompletion
  - @AfterCompletion
- Events
  - @Observes(during=IN\_PROGRESS)
  - @Observes(during=BEFORE\_COMPLETION)
  - @Observes(during=AFTER\_COMPLETION)
    - @Observes(during=AFTER\_SUCCESS)
    - @Observes(during=AFTER\_FAILURE)



```
@Stateful
public class TransactionListenerBean {

    // any number of other methods and fields...
    // invoke me and I will listen on this transaction

    @AfterBegin
    public void joiningTransactionInProgress() {

        // participate in the transaction at will
    }

    @BeforeCompletion
    public void nearlyDone() {

        // last chance to do something
        // in the transaction
    }

    @AfterCompletion
    public void done(boolean succeeded){

        if (succeeded) {
            // Yay!!
        } else {
            // Boo!!
        }
    }
}
```

# “Undo” aware Resources

- DataSource
- EntityManager
- Sending JMS Messages
- Receiving JMS Messages
- Timers & @Schedule
- (more via Java EE connectors)

*[Some restrictions apply. Offer void where prohibited. Container-provided resources required. Must be 18 years or older to apply]*

# More specifically

```
@PersistenceContext // undo entity create, update, deletes  
private javax.persistence.EntityManager entityManager;
```

```
@Resource // you can undo executed statements  
private javax.sql.DataSource dataSource;
```

```
@Resource // you can undo a send or receive  
private javax.jms.ConnectionFactory jmsConnectionFactory;
```

```
@Resource // you can undo the scheduling or execution of work  
private javax.ejb.TimerService timerService;
```

```
@Resource // you can observe events in a transactional way  
private javax.enterprise.event.Event event;
```

How does it work?

**TOP SECRET**  
(authorized personnel only)



This is not a pipe

# Not the real thing

```
@PersistenceContext  
private EntityManager thisIsNotAnEntityManager;
```

```
@Resource  
private DataSource thisIsNotADataSource;
```

# The EntityManager You Get



```
public class WhatYouGet implements EntityManager {

    @Override
    public void persist(Object entity) {
        getTheRealDeal().persist(entity);
    }

    @Override
    public <T> T merge(T entity) {
        return getTheRealDeal().merge(entity);
    }

    @Override
    public void remove(Object entity) {
        getTheRealDeal().remove(entity);
    }

    @Override
    public void close() {
        // ignore
    }

    @Override
    public EntityTransaction getTransaction() {
        throw new IllegalStateException();
    }

    //...
}
```

# continued...

```
private final TransactionSynchronizationRegistry transactionRegistry;

private final EntityManagerFactory entityManagerFactory;

public EntityManager getTheRealDeal() {

    if (!isTransactionInProgress()) throw new TransactionRequiredException();

    // Do we have one?
    final Object resource = transactionRegistry.getResource("some key");
    EntityManager realEntityManager = (EntityManager) resource;

    // Create one
    if (realEntityManager == null) {
        realEntityManager = entityManagerFactory.createEntityManager();
        transactionRegistry.putResource(realEntityManager, "some key");
    }

    return realEntityManager;
}
```

# The DataSource You Get

```
public class WhatYouGet implements DataSource {  
  
    private final TransactionSynchronizationRegistry transactionRegistry;  
  
    @Override  
    public Connection getConnection() throws SQLException {  
  
        if (isTransactionInProgress()) {  
            return getConnectionFromTransaction();  
        } else {  
            return getConnectionFromPool();  
        }  
    }  
  
    private Connection getConnectionFromTransaction() {  
        // Do we have one?  
        final Object resource = transactionRegistry.getResource("some key");  
        Connection realConnection = (Connection) resource;  
  
        if (realConnection == null) {  
            realConnection = getConnectionFromPool();  
            transactionRegistry.putResource(realConnection, "some key");  
        }  
  
        return realConnection;  
    }  
}
```



better than a pipe

# ...better than the real thing

```
@Resource  
private DataSource uberDataSource;  
  
@PersistenceContext  
private EntityManager uberEntityManager;
```

- thread-safe
- undo-aware
- leak-free

All boils down to...

# “hashmap”

```
package javax.transaction;

public interface TransactionSynchronizationRegistry {

    Object getResource(Object key);

    void putResource(Object key, Object value);

    void registerInterposedSynchronization(Synchronization sync);

    //.. other methods
}
```

# “listeners”

```
package javax.transaction;

public interface Synchronization {

    void beforeCompletion();

    void afterCompletion(int status);

}
```



...but



# Won't work with any of this

- DataSource
  - DriverManager.getConnection(...)
  - new FooDataSource()
  - autocommit = true
- EntityManager
  - PersistenceProvider.createEntityManag...
  - EntityManagerFactory.createEntityManager
  - EntityTransaction



**Avoid Do It Yourself**

# Rule

- If you didn't get it from the Container
  - it isn't usable with UserTransaction
  - connections may leak
  - memory may leak
- Even then there are some BIG notes
  - EntityManagerFactory
    - not usable with UserTransaction
  - `<persistence-unit=RESOURCE_LOCAL>`
    - not usable with UserTransaction



# Common Migration Issues & Mistakes

# Wrong

```
@Resource
private UserTransaction userTransaction;

public void wrong() throws Exception {

    userTransaction.begin();
    try {
        org.apache.derby.jdbc.ClientDataSource dataSource
            = new org.apache.derby.jdbc.ClientDataSource();

        dataSource.setServerName("my_derby_database_server");
        dataSource.setDatabaseName("my_derby_database_name");

        final Connection connection = dataSource.getConnection();
        //.. use the connection

        userTransaction.commit();
    } catch (Throwable t) {
        userTransaction.rollback();
    }
}
```

# Wrong

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence" version="1.0">
  <persistence-unit name="foo-unit">
    <properties>
      <property name="openjpa.ConnectionDriverName" value="org.hsqldb.jdbcDriver" />
      <property name="openjpa.ConnectionURL" value="jdbc:hsqldb:mem:hsqldb" />
      <property name="openjpa.ConnectionUserName" value="joe" />
      <property name="openjpa.ConnectionPassword" value="cool" />
    </properties>
  </persistence-unit>
</persistence>
```



# Wrong

```
@Resource
private UserTransaction userTransaction;

public void wrong() throws Exception {

    userTransaction.begin();
    try {

        final EntityManagerFactory entityManagerFactory =
            Persistence.createEntityManagerFactory("foo-unit");

        final EntityManager entityManager =
            entityManagerFactory.createEntityManager();

        entityManager.persist(new Movie());

        userTransaction.commit();
    } catch (Throwable t) {
        userTransaction.rollback();
    }
}
```

# Still Wrong

```
@Resource
private UserTransaction userTransaction;

@PersistenceUnit(unitName = "foo-unit")
private EntityManagerFactory entityManagerFactory;

public void stillWrong() throws Exception {

    userTransaction.begin();
    try {
        final EntityManager entityManager =
            entityManagerFactory.createEntityManager();

        entityManager.persist(new Movie());

        userTransaction.commit();
    } catch (Throwable t) {
        userTransaction.rollback();
    }
}
```



The worst of them all

# this...

```
public void doSomething() throws Exception {  
    final EntityManagerFactory entityManagerFactory =  
        Persistence.createEntityManagerFactory("foo-unit");  
  
    final EntityManager entityManager =  
        entityManagerFactory.createEntityManager();  
  
    entityManager.persist(new Movie());  
}
```

or this...

```
@PersistenceUnit(unitName = "foo-unit")
private EntityManagerFactory entityManagerFactory;

public void doSomething() throws Exception {

    final EntityManager entityManager =
        entityManagerFactory.createEntityManager();

    entityManager.persist(new Movie());

}
```

# with THIS!

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             version="1.0">
  <persistence-unit name="foo-unit">
    <non-jta-data-source>fooDataSource</non-jta-data-source>
    <properties>
      <!-- some properties-->
    </properties>
  </persistence-unit>
</persistence>
```

# darn defaults!

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  version="1.0">
  <persistence-unit name="foo-unit" transaction-type="JTA">
    <non-jta-data-source>fooDataSource</non-jta-data-source>
    <properties>
      <!-- some properties-->
    </properties>
  </persistence-unit>
</persistence>
```

the default



# There we go

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             version="1.0">
  <persistence-unit name="foo-unit" transaction-type="RESOURCE_LOCAL">
    <non-jta-data-source>fooDataSource</non-jta-data-source>
    <properties>
      <!-- some properties-->
    </properties>
  </persistence-unit>
</persistence>
```

# JPA Rules

- RESOURCE\_LOCAL
  - Persistence.createEntityManagerFactory
  - @PersistenceUnit EntityManagerFactory
  - <non-jta-data-source>
- JTA
  - @PersistenceContext EntityManager
  - <jta-data-source>
  - <non-jta-data-source>
- No other combinations are valid

# Good

- Valid RESOURCE\_LOCAL usage

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
              version="1.0">
  <persistence-unit name="foo-unit" transaction-type="RESOURCE_LOCAL">
    <non-jta-data-source>fooDataSource</non-jta-data-source>
  </persistence-unit>
</persistence>
```

```
public class ValidUserManagedEntityManager {
    public void thisWillWork() throws Exception {
        final EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("foo-unit");
        final EntityManager entityManager = emf.createEntityManager();
        // ...
    }
}
```

# Better

- Valid RESOURCE\_LOCAL usage

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             version="1.0">
  <persistence-unit name="foo-unit" transaction-type="RESOURCE_LOCAL">
    <non-jta-data-source>fooDataSource</non-jta-data-source>
  </persistence-unit>
</persistence>
```

```
public class ValidUserManagedEntityManager {
  @PersistenceUnit(unitName = "foo-unit")
  private EntityManagerFactory emf;

  public void thisWillWork() throws Exception {
    final EntityManager entityManager = emf.createEntityManager();

    // ...
  }
}
```

# Best

- Valid JTA usage

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
              version="1.0">
  <persistence-unit name="foo-unit">
    <jta-data-source>fooJtaDataSource</jta-data-source>
    <non-jta-data-source>fooDataSource</non-jta-data-source>
  </persistence-unit>
</persistence>
```

```
public class ValidContainerManagedEntityManager {
  @PersistenceContext(unitName = "foo-unit")
  private EntityManager entityManager;

  public void thisWillWork() throws Exception {
    // use the EntityManager in any transaction...
  }
}
```

# DataSource

- Declare
  - in `tomee.xml`
  - in `META-INF/resources.xml`
  - via `@DataSourceDefinition`
- Retrieve
  - `@Resource`
  - `<resource-ref>`
- Undo-aware and pooled

# Declaring

- <tomee-home>/conf/tomee.xml

```
<tomee>  
  <Resource id="fooDataSource" type="DataSource">  
    JdbcDriver = org.apache.derby.jdbc.ClientDriver  
    JdbcUrl    = jdbc:derby://localhost:1527/derbydb;create=true  
    Username   = joe  
    Password  = cool  
  </Resource>  
</tomee>
```

- WEB-INF/resources.xml

```
<resources>  
  <Resource id="fooDataSource" type="DataSource">  
    JdbcDriver = org.apache.derby.jdbc.ClientDriver  
    JdbcUrl    = jdbc:derby://localhost:1527/derbydb;create=true  
    Username   = joe  
    Password  = cool  
  </Resource>  
</resources>
```

# Recap



# Recap

- Use if you need “undo”
- Transactions not heavy
  - Thread safe management
  - Help prevent Leaks
  - Enhance programming
- More than a TransactionManager
  - TransactionManagers do nothing alone
  - Resources must cooperate
- Don't Do It Yourself
  - Use container-provided resources
  - no need to re-invent the wheel

# Recap

- Components with Transaction support
  - Servlets
  - JSP
  - JSF Managed Beans
  - CDI
  - EJB
  - JAX-RS REST Services
  - JAX-WS Web Services

# Recap

- Transactional Resources
  - DataSource
  - EntityManager
  - Sending JMS Messages
  - Receiving JMS Messages
  - Timers & @Schedule
  - (more via Java EE connectors)

Questions?

Thank You!

@dblevins

<http://tomee.apache.org>