



Java Code Geeks

JAVA 2 JAVA DEVELOPERS RESOURCE CENTER

Are you ready to get 100 high authority backlinks for free?

Submission is 100% free and take less than a minute, no registration required!

[Start Submit](#)
[Java](#) | [Android](#) | [JVM Languages](#) | [Software Development](#) | [Agile](#) | [DevOps](#) | [Communications](#) | [Career](#) | [Misc](#) | [Meta JCG](#)

Core Java | Java | Spring JTA multiple resource transactions in Tomcat with Atomikos example

Febr

Enterprise Java | [Iora Fragkouli](#)

Desktop Java | graduated from Computer Engineering and Informatics Department in the University of Patras. She also holds a in Economics from the National and Technical University of Athens. During her studies she has been involved with a large number of projects ranging from programming and software engineering to telecommunications, hardware design and analysis.



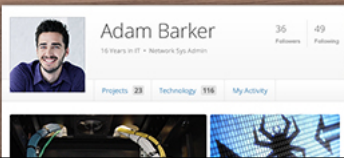
Advertise Here

Advertise H

Spring JTA multiple resource transactions in Tomcat with Atomikos example

by Theodora Fragkouli on July 3rd, 2013 | Filed in: Enterprise Java | Tags: Apache Tomcat, Atomikos, JTA, Spring

Spotlight your tech prowess with a Spiceworks profile



In this tutorial we shall show you how to implement JTA multiple resource transactions in a [Tomcat](#) server, using [Atomikos Transaction Manager](#). Atomikos transaction manager provides support for distributed transactions. These are multi-phased transactions, often using multiple databases, that must be committed in a coordinated way. The distributed transactions are described by the XA standard. XA governs how a transaction manager (such as Atomikos) can tell a database what work is going on as part of what transaction, and how to conduct the two-phase commit (2PC) protocol at the end of each transaction.

Here, we will create simple Entity classes mapped to two different databases and we will try to persist objects of the classes to the databases using one distributed transaction. We will also see what happens when one of the underlying transactions

rollbacks.

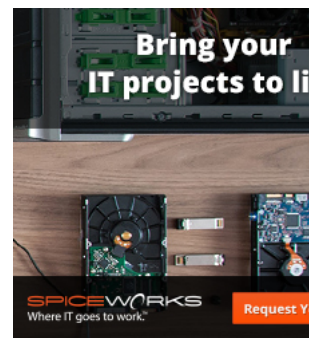
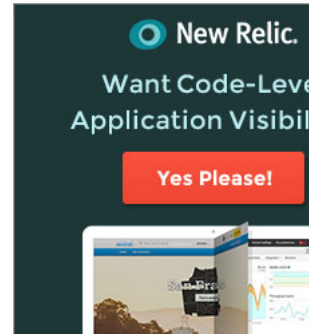
Our preferred development environment is [Eclipse](#). We are using Eclipse Juno (4.2) version, along with [Maven](#) Integration plugin version 3.1.0. You can download Eclipse from [here](#) and Maven Plugin for Eclipse from [here](#). The installation of Maven plugin for Eclipse is out of the scope of this tutorial and will not be discussed. We are also using Spring version 3.2.3 and the JDK 7_u_21.

[Tomcat 7](#) is the application server used. Hibernate version is 4.1.9, and the database used in the example is MySQL Database Server 5.6.

Let's begin,

1. Create a new Maven project

Go to File -> Project -> Maven -> Maven Project.



Newsletter

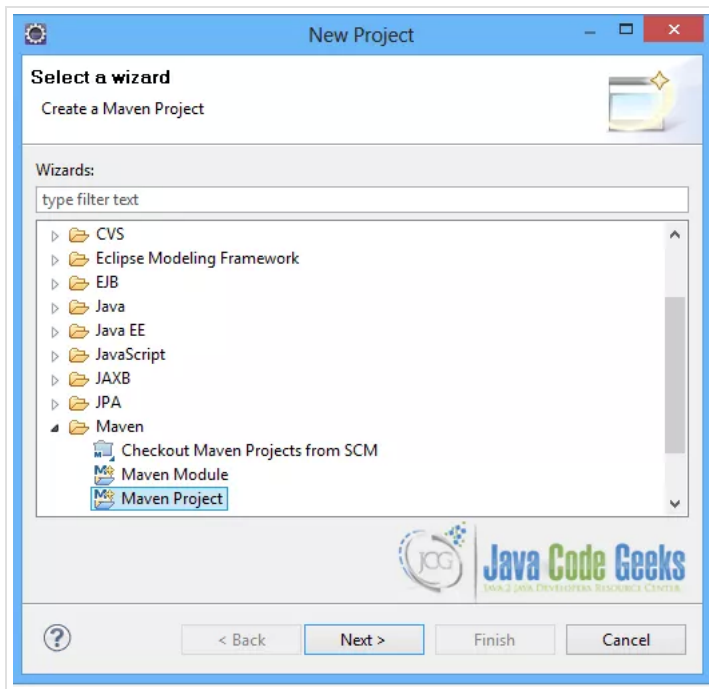


new e-books, published by Java C
their JCG partners for your reading

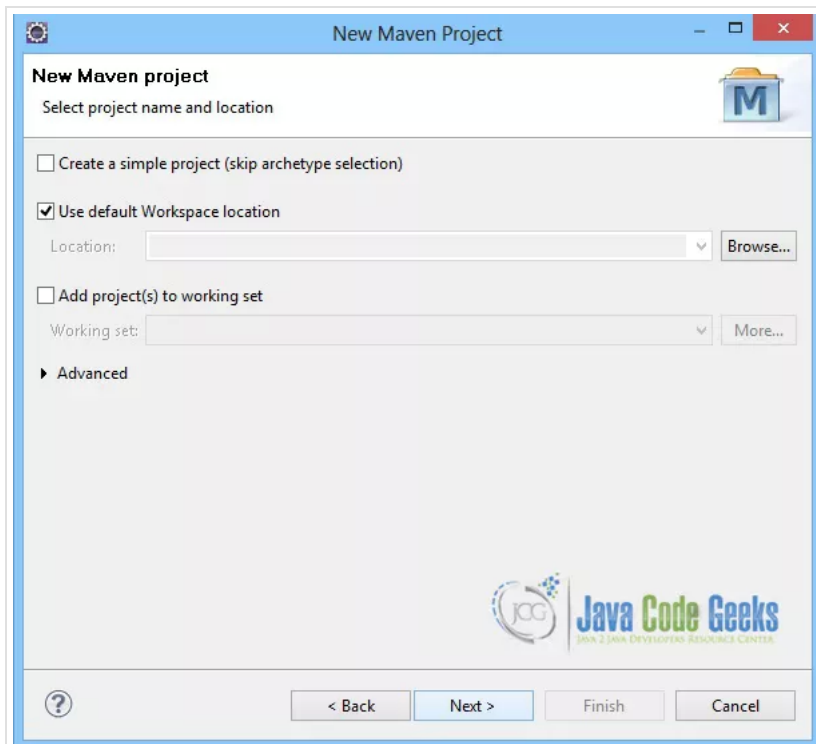
Join Us



If you have a blog with unique and i
content then you should check out
partners program. You can also be
for Java Code Geeks and hone you
addition to utilizing our [revenue sh](#)
[monetize](#) your technical writing!



In the “Select project name and location” page of the wizard, make sure that “Create a simple project (skip archetype selection)” option is **unchecked**, hit “Next” to continue with default values.



Here the maven archetype for creating a web application must be added. Click on “**Add Archetype**” and add the archetype. Set the “Archetype Group Id” variable to “org.apache.maven.archetypes”, the “Archetype artifact Id” variable to “maven-archetype-webapp” and the “Archetype Version” to “1.0”. Click on “**OK**” to continue.

Carrer Opportunities

Scientist-Cognitive Systems (FULL-TIME) February 5th, 2014

Software Developer, 2 (FULL-TIME) 4th, 2014

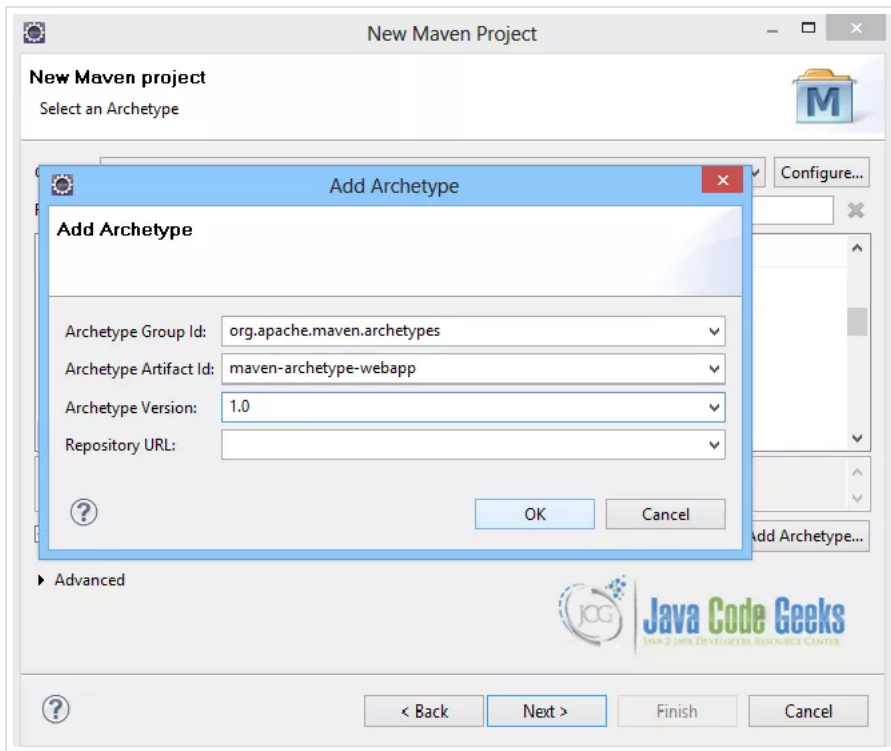
Junior Java Engineer (FULL-TIME) 2014

Software Development Engineer (FULL-TIME) February 4th, 2014

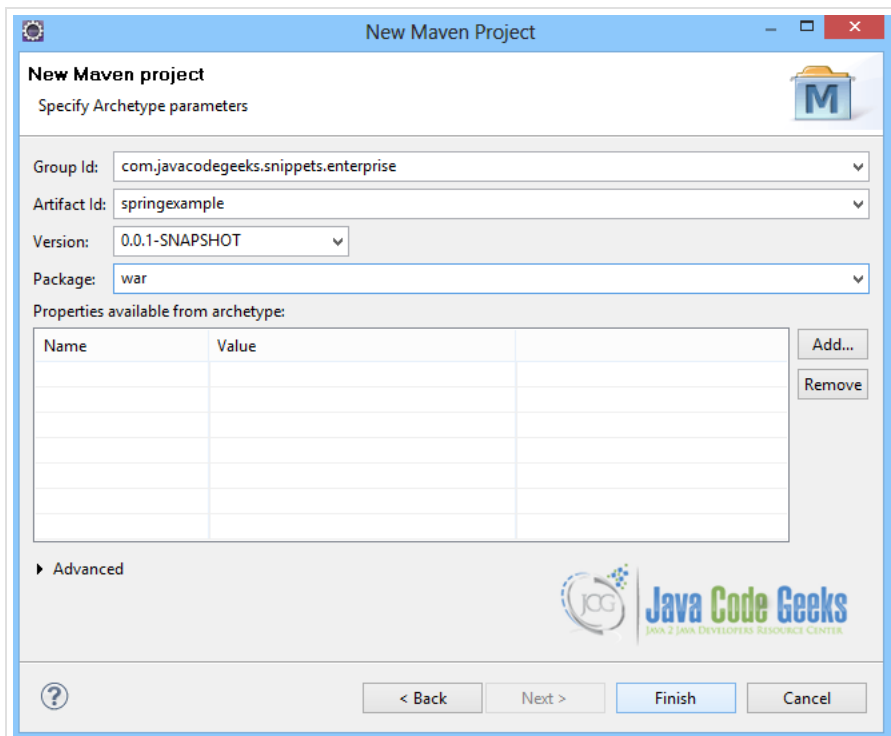
Enterprise Java Software Developer (FULL-TIME) February 4th, 2014

Tags

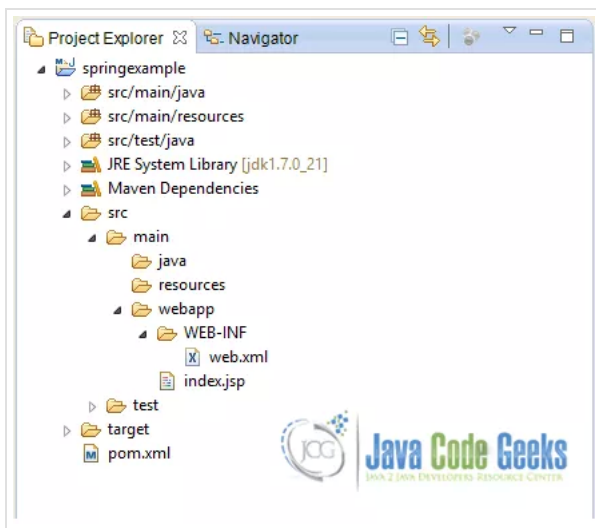
Akka Android Tutorial Apache Camel Apache Maven Apache Tomcat Cloud Concurrency Design Patterns Eclipse Git Grails IDE Interview Java 7 JavaFX JavaOne JAXB JBoss Hibernate JPA JSF JVM Logging MongoDB Oracle GlassFish Performance Project Management RESTful Web Services Security Spring MVC Spring Security Spring T



In the "Enter an artifact id" page of the wizard, you can define the name and main package of your project. Set the "Group Id" variable to "com.javacodegeeks.snippets.enterprise" and the "Artifact Id" variable to "springexample". The aforementioned selections compose the main project package as "com.javacodegeeks.snippets.enterprise.springexample" and the project name as "springexample". Set the "Package" variable to "war", so that a war file will be created to be deployed to tomcat server. Hit "Finish" to exit the wizard and to create your project.



The Maven project structure is shown below:



It consists of the following folders:

- /src/main/java folder, that contains source files for the dynamic content of the application,
- /src/test/java folder contains all source files for unit tests,
- /src/main/resources folder contains configurations files,
- /target folder contains the compiled and packaged deliverables,
- /src/main/resources/webapp/WEB-INF folder contains the deployment descriptors for the Web application ,
- the pom.xml is the project object model (POM) file. The single file that contains all project related configuration.

2. Add Spring 3.2.3 dependency

- Locate the "Properties" section at the "Overview" page of the POM editor and perform the following changes:
Create a new property with name **org.springframework.version** and value **3.2.3.RELEASE**.
- Navigate to the "Dependencies" page of the POM editor and create the following dependencies (you should fill the "GroupId", "Artifact Id" and "Version" fields of the "Dependency Details" section at that page):
Group Id : **org.springframework** Artifact Id : **spring-web** Version : **\${org.springframework.version}**

Alternatively, you can add the Spring dependencies in Maven's `pom.xml` file, by directly editing it at the "Pom.xml" page of the POM editor, as shown below:

`pom.xml`:

```

01 <project xmlns="http://maven.apache.org/POM/4.0.0"; xmlns:xsi="http://www.w3.org/2001/XMLSchema-
02   instance"
03   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
04   4.0.0.xsd">
05   <modelVersion>4.0.0</modelVersion>
06   <groupId>com.javacodegeeks.snippets.enterprise</groupId>
07   <artifactId>springexample</artifactId>
08   <version>0.0.1-SNAPSHOT</version>
09   <dependencies>
10     <dependency>
11       <groupId>org.springframework</groupId>
12       <artifactId>spring-core</artifactId>
13       <version>${spring.version}</version>
14     </dependency>
15     <dependency>
16       <groupId>org.springframework</groupId>
17       <artifactId>spring-context</artifactId>
18       <version>${spring.version}</version>
19     </dependency>
20   </dependencies>
21   <properties>
22     <spring.version>3.2.3.RELEASE</spring.version>
23   </properties>
24 </project>

```

As you can see Maven manages library dependencies declaratively. A local repository is created (by default under `{user_home}/.m2` folder) and all required libraries are downloaded and placed there from public repositories. Furthermore intra – library dependencies are automatically resolved and manipulated.

3. Add all required dependencies

All dependencies needed to set up atomikos transaction manager are set here.

`pom.xml`:

```

001 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
002   instance"
003   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

```

```
003 <modelVersion>4.0.0</modelVersion>
004 <groupId>com.javacodegeeks.snippets.enterprise</groupId>
005 <artifactId>springexample</artifactId>
006 <packaging>war</packaging>
007 <version>0.0.1</version>
008 <name>springexample Maven Webapp</name>
009 <url>http://maven.apache.org</url>
010 <build>
011 <finalName>springexample</finalName>
012 </build>
013 <dependencies>
014 <dependency>
015 <groupId>org.springframework</groupId>
016 <artifactId>spring-core</artifactId>
017 <version>${spring.version}</version>
018 </dependency>
019
020 <dependency>
021 <groupId>org.springframework</groupId>
022 <artifactId>spring-context</artifactId>
023 <version>${spring.version}</version>
024 </dependency>
025
026 <dependency>
027 <groupId>org.springframework</groupId>
028 <artifactId>spring-tx</artifactId>
029 <version>${spring.version}</version>
030 </dependency>
031
032 <dependency>
033 <groupId>org.springframework</groupId>
034 <artifactId>spring-orm</artifactId>
035 <version>${spring.version}</version>
036 </dependency>
037
038 <dependency>
039 <groupId>org.springframework</groupId>
040 <artifactId>spring-web</artifactId>
041 <version>${spring.version}</version>
042 </dependency>
043
044 <dependency>
045 <groupId>org.springframework</groupId>
046 <artifactId>spring-webmvc</artifactId>
047 <version>${spring.version}</version>
048 </dependency>
049
050 <dependency>
051 <groupId>org.hibernate</groupId>
052 <artifactId>hibernate-entitymanager</artifactId>
053 <version>${hibernate.version}</version>
054 <exclusions>
055 <exclusion>
056 <groupId>cglib</groupId>
057 <artifactId>cglib</artifactId>
058 </exclusion>
059 <exclusion>
060 <groupId>dom4j</groupId>
061 <artifactId>dom4j</artifactId>
062 </exclusion>
063 </exclusions>
064 </dependency>
065
066 <dependency>
067 <groupId>javax.servlet</groupId>
068 <artifactId>javax.servlet-api</artifactId>
069 <version>3.0.1</version>
070 <scope>provided</scope>
071 </dependency>
072
073 <dependency>
074 <groupId>com.atomikos</groupId>
075 <artifactId>transactions-jta</artifactId>
076 <version>${atomikos.version}</version>
077 </dependency>
078
079 <dependency>
080 <groupId>com.atomikos</groupId>
081 <artifactId>transactions-jdbc</artifactId>
082 <version>${atomikos.version}</version>
083 </dependency>
084
085 <dependency>
086 <groupId>com.atomikos</groupId>
087 <artifactId>transactions-hibernate3</artifactId>
088 <version>${atomikos.version}</version>
089 <exclusions>
090 <exclusion>
091 <artifactId>hibernate</artifactId>
092 <groupId>org.hibernate</groupId>
093 </exclusion>
094 </exclusions>
095 </dependency>
096
097 <dependency>
098 <groupId>dom4j</groupId>
099 <artifactId>dom4j</artifactId>
100 <version>1.6.1</version>
101 </dependency>
102
103 <dependency>
104 <groupId>mysql</groupId>
105 <artifactId>mysql-connector-java</artifactId>
```

```

106     <version>5.1.25</version>
107 </dependency>
108 </dependencies>
109
110 <properties>
111 <spring.version>3.2.3.RELEASE</spring.version>
112 <hibernate.version>4.1.9.Final</hibernate.version>
113 <atomikos.version>3.8.0</atomikos.version>
114 </properties>
115
116 </project>

```

4. Create the Entity classes

EmployeeA.java and EmployeeB.java are the Entity classes. They use the `javax.persistence` annotations to be mapped to a table, `EMPLOYEEA` and `EMPLOYEEB` in different databases. In particular, the `@Entity` annotation specifies that each class is an entity. The `@Table` annotation specifies the primary table for the annotated entity. The `@Column` annotation is used to specify a mapped column for the persistent field, whereas the `@Id` annotation specifies the primary key field of each entity.

EmployeeA.java

```

01 package com.javacodegeeks.snippets.enterprise.model;
02
03 import javax.persistence.Column;
04 import javax.persistence.Entity;
05 import javax.persistence.Id;
06 import javax.persistence.Table;
07
08 @Entity
09 @Table(name = "EMPLOYEEA")
10 public class EmployeeA {
11
12     @Id
13     @Column(name = "ID", nullable = false)
14     private String id;
15
16     @Column(name = "NAME", nullable = false)
17     private String name;
18
19     @Column(name = "AGE", nullable = false)
20     private long age;
21
22     public EmployeeA() {
23     }
24
25     public String getId() {
26         return id;
27     }
28
29     public void setId(String id) {
30         this.id = id;
31     }
32
33     public String getName() {
34         return name;
35     }
36
37     public void setName(String name) {
38         this.name = name;
39     }
40
41     public long getAge() {
42         return age;
43     }
44
45     public void setAge(long age) {
46         this.age = age;
47     }
48
49 }

```

EmployeeB.java

```

01 package com.javacodegeeks.snippets.enterprise.model;
02
03 import javax.persistence.Column;
04 import javax.persistence.Entity;
05 import javax.persistence.Id;
06 import javax.persistence.Table;
07
08 @Entity
09 @Table(name = "EMPLOYEEB")
10 public class EmployeeB {
11
12     @Id
13     @Column(name = "ID", nullable = false)
14     private String id;
15
16     @Column(name = "NAME", nullable = false)
17     private String name;
18
19     @Column(name = "AGE", nullable = false)
20     private long age;
21
22     public EmployeeB() {
23     }
24
25     public String getId() {

```

```

26         return id;
27     }
28
29     public void setId(String id) {
30         this.id = id;
31     }
32
33     public String getName() {
34         return name;
35     }
36
37     public void setName(String name) {
38         this.name = name;
39     }
40
41     public long getAge() {
42         return age;
43     }
44
45     public void setAge(long age) {
46         this.age = age;
47     }
48
49 }

```

5. Create the DAO classes

The Data Access Objects implemented are the `EmployeeADAOImpl.java` and `EmployeeBDAOImpl.java` classes. They are annotated with the `@Service` annotation, dictating that they are Spring Beans and thus allowing Spring to auto-detect them. They both use the `javax.persistence.EntityManager` to interact with the databases.

An `EntityManager` instance is associated with a persistence context. A persistence context is a set of entity instances in which for any persistent entity identity there is a unique entity instance. Within the persistence context, the entity instances and their lifecycle are managed. The `EntityManager` API is used to create and remove persistent entity instances, to find entities by their primary key, and to query over entities. The `EntityManager` is configured in `persistence.xml` file, that is described in paragraph 8.1.

The set of entities that can be managed by a given `EntityManager` instance is defined by a persistence unit. A persistence unit defines the set of all classes that are related or grouped by the application, and which must be colocated in their mapping to a single database.

The `EntityManager` is injected in each DAO with the `@PersistenceContext` annotation, where the name of each persistence unit is set, as defined in the `persistence.xml` file.

A basic persist method is implemented in both DAOs, using the `persist(Object entity)` API method of `EntityManager` to create an object to the database.

The DAOs and their interfaces are shown below:

`EmployeeADAO.java`

```

1 package com.javacodegeeks.snippets.enterprise.dao;
2
3 import com.javacodegeeks.snippets.enterprise.model.EmployeeA;
4
5 public interface EmployeeADAO {
6
7     void persistEmployee(EmployeeA employee);
8 }

```

`EmployeeADAO Impl.java`

```

01 package com.javacodegeeks.snippets.enterprise.dao;
02
03 import javax.persistence.EntityManager;
04 import javax.persistence.PersistenceContext;
05
06 import org.springframework.stereotype.Service;
07
08 import com.javacodegeeks.snippets.enterprise.model.EmployeeA;
09
10 @Service
11 public class EmployeeADAOImpl implements EmployeeADAO {
12
13     @PersistenceContext(unitName="PersistenceUnitA")
14     private EntityManager entityManager;
15
16     public void persistEmployee(EmployeeA employee) {
17         entityManager.persist(employee);
18     }
19
20 }

```

`EmployeeBDAO.java`

```

1 package com.javacodegeeks.snippets.enterprise.dao;
2
3 import com.javacodegeeks.snippets.enterprise.model.EmployeeB;
4
5 public interface EmployeeBDAO {
6
7     void persistEmployee(EmployeeB employee) throws Exception;
8 }

```



```
9 | }
```

EmployeeBDAO Impl.java

```
01 | package com.javacodegeeks.snippets.enterprise.dao;
02 |
03 |
04 | import javax.persistence.EntityManager;
05 | import javax.persistence.PersistenceContext;
06 |
07 | import org.springframework.stereotype.Service;
08 |
09 | import com.javacodegeeks.snippets.enterprise.model.EmployeeB;
10 |
11 | @Service
12 | public class EmployeeBDAOImpl implements EmployeeBDAO {
13 |
14 |     @PersistenceContext(unitName="PersistenceUnitB")
15 |     private EntityManager entityManager;
16 |
17 |     public void persistEmployee(EmployeeB employee) throws Exception {
18 |         entityManager.persist(employee);
19 |         // throw new Exception();
20 |     }
21 | }
```

6. Create the Service class

The `EmployeeADAOImpl.java` and `EmployeeBDAOImpl.java` classes are injected in the `EmployeeServiceImpl.java` class. Thus, in the `persistEmployees(EmployeeA employeeA, EmployeeB employeeB)` method implemented here, the DAOs' methods are invoked to perform the basic interaction with the database. The `EmployeeServiceImpl.java` class is also annotated with the `@Service` annotation, dictating that it is a Spring Bean and thus allowing Spring to auto-detect it.

The `@Transactional` annotation is placed before the method, to denote that a transaction is created when the method is invoked. The transaction is a global container managed transaction and will be configured in Spring configuration file.

EmployeeService.java

```
01 | package com.javacodegeeks.snippets.enterprise.service;
02 |
03 | import com.javacodegeeks.snippets.enterprise.model.EmployeeA;
04 | import com.javacodegeeks.snippets.enterprise.model.EmployeeB;
05 |
06 | public interface EmployeeService {
07 |
08 |     void persistEmployees(EmployeeA employeeA, EmployeeB employeeB) throws Exception;
09 |
10 | }
```

EmployeeServiceImpl.java

```
01 | package com.javacodegeeks.snippets.enterprise.service;
02 |
03 | import org.springframework.beans.factory.annotation.Autowired;
04 | import org.springframework.stereotype.Service;
05 | import org.springframework.transaction.annotation.Transactional;
06 |
07 | import com.javacodegeeks.snippets.enterprise.dao.EmployeeADAO;
08 | import com.javacodegeeks.snippets.enterprise.dao.EmployeeBDAO;
09 | import com.javacodegeeks.snippets.enterprise.model.EmployeeA;
10 | import com.javacodegeeks.snippets.enterprise.model.EmployeeB;
11 |
12 | @Service("employeeService")
13 | public class EmployeeServiceImpl implements EmployeeService{
14 |
15 |     @Autowired
16 |     EmployeeADAO employeeADAO;
17 |
18 |     @Autowired
19 |     EmployeeBDAO employeeBDAO;
20 |
21 |     @Transactional(rollbackFor=Exception.class)
22 |     public void persistEmployees(EmployeeA employeeA, EmployeeB employeeB) throws Exception {
23 |         System.out.println("Persist A");
24 |         employeeADAO.persistEmployee(employeeA);
25 |         System.out.println("Persist A OK - persist B");
26 |         employeeBDAO.persistEmployee(employeeB);
27 |         System.out.println("Persist B okk");
28 |     }
29 |
30 | }
```

7. Create a servlet to run the application

The `AppServlet.java` class is a simple servlet, that implements the `org.springframework.web.HttpRequestHandler` and overrides its `handleRequest(HttpServletRequest req, HttpServletResponse resp)` API method. The `EmployeeService` is injected here, via the `@Autowire` annotation. It is used in the `handleRequest(HttpServletRequest req, HttpServletResponse resp)` API method to persist a new `EmployeeA` and a new `EmployeeB` object. The method also returns a success message if the method returns successfully and rollback message if the method throws an exception.

AppServlet.java

```
01 | package com.javacodegeeks.snippets.enterprise.servlet;
```



```

02
03 import java.io.IOException;
04 import java.io.PrintWriter;
05
06 import javax.servlet.ServletException;
07 import javax.servlet.http.HttpServletRequest;
08 import javax.servlet.http.HttpServletResponse;
09
10 import org.springframework.beans.factory.annotation.Autowired;
11 import org.springframework.stereotype.Component;
12 import org.springframework.web.HttpRequestHandler;
13
14 import com.javacodegeeks.snippets.enterprise.model.EmployeeA;
15 import com.javacodegeeks.snippets.enterprise.model.EmployeeB;
16 import com.javacodegeeks.snippets.enterprise.service.EmployeeService;
17
18 @Component("appServlet")
19 public class AppServlet implements HttpRequestHandler {
20
21     @Autowired
22     private EmployeeService employeeService;
23
24     public void handleRequest(HttpServletRequest req, HttpServletResponse resp)
25         throws ServletException, IOException {
26         EmployeeA em1 = new EmployeeA();
27         em1.setId("123");
28         em1.setName("John");
29         em1.setAge(35);
30         EmployeeB em2 = new EmployeeB();
31         em2.setId("123");
32         em2.setName("Mary");
33         em2.setAge(31);
34
35         try {
36             employeeService.persistEmployees(em1, em2);
37             resp.setContentType("text/html");
38             PrintWriter out = resp.getWriter();
39             out.println("<html>");
40             out.println("<head>");
41             out.println("<title>Hello World!</title>");
42             out.println("</head>");
43             out.println("<body>");
44             out.println("<h1>Java Code Geeks </h1>");
45             out.println("<h2>Both employees are inserted!</h2>");
46             out.println("</body>");
47             out.println("</html>");
48         } catch (Exception e) {
49             resp.setContentType("text/html");
50             PrintWriter out = resp.getWriter();
51             out.println("<html>");
52             out.println("<head>");
53             out.println("<title>Hello World!</title>");
54             out.println("</head>");
55             out.println("<body>");
56             out.println("<h1>Java Code Geeks </h1>");
57             out.println("<h2>Transaction Rollback!</h2>");
58             out.println("</body>");
59             out.println("</html>");
60             e.printStackTrace();
61         }
62     }
63 }

```

8. Configure the application

8.1 Configure the persistence units

As mentioned above, the `entityManager` and the persistence unit associated with it for every database is configured in `persistence.xml` file. Here we define two persistence units. In every `persistence-unit` element, we define the entity class associated to the persistence-unit. The `hibernate.transaction.manager_lookup_class` property is set to `com.atomikos.icatch.jta.hibernate3.TransactionManagerLookup`. The `hibernate.transaction.factory_class` property is set to `org.hibernate.transaction.CMTTransactionFactory`.

[persistence.xml](#)

```

01 <?xml version="1.0" encoding="UTF-8"?>
02 <persistence version="2.0"
03     xmlns="http://java.sun.com/xml/ns/persistence"
04     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
05     xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
06         http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
07
08     <persistence-unit name="PersistenceUnitA" transaction-type="JTA">
09         <class>com.javacodegeeks.snippets.enterprise.model.EmployeeA</class>
10         <properties>
11             <property name="hibernate.transaction.manager_lookup_class"
12                 value="com.atomikos.icatch.jta.hibernate3.TransactionManagerLookup" />
13             <property name="hibernate.transaction.factory_class"
14                 value="org.hibernate.transaction.CMTTransactionFactory" />
15         </properties>
16     </persistence-unit>
17
18     <persistence-unit name="PersistenceUnitB" transaction-type="JTA">
19         <class>com.javacodegeeks.snippets.enterprise.model.EmployeeB</class>
20         <properties>
21             <property name="hibernate.transaction.manager_lookup_class"
22                 value="com.atomikos.icatch.jta.hibernate3.TransactionManagerLookup" />
23             <property name="hibernate.transaction.factory_class"
24                 value="org.hibernate.transaction.CMTTransactionFactory" />
25         </properties>
26     </persistence-unit>
27 </persistence>

```

```

25     value="org.hibernate.transaction.CMTTransactionFactory" />
26   </properties>
27 </persistence-unit>
28
29 </persistence>

```

8.2 Configure Spring container

The `applicationContext.xml` file is the configuration file of Spring.

The `<context:component-scan/>` element is used to set the package that contains all classes that the container must scan to detect the Spring beans.

The `<tx:annotation-driven/>` element is also used so that Spring is `@Transactional-aware` and can detect the `@Transactional` annotations to configure the appropriate beans with transactional behavior.

The `<jta-transaction-manager/>` element is used to detect the underlying server and choose the transaction manager available for the platform.

In `dataSourceA` and `dataSourceB` beans we define the datasources. The `com.atomikos.jdbc.AtomikosDataSourceBean` is the class set here. It uses Atomikos JTA-enabled connection pooling. It has two properties to configure. The `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource` class is set to the `xaDataSourceClass` property, whereas in the `xaProperties` we can set the properties (name,value pairs) to configure the `XADataSource`.

In `entityManagerFactoryA` and `entityManagerFactoryB` beans we set the `org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean` class. It is a `FactoryBean` that creates a `JPA EntityManagerFactory` according to JPA's standard container bootstrap contract. We can set the `persistence.xml` location in its `persistenceXmlLocation` property. We can set the name of the persistence unit used to create this `EntityManagerFactory`, in `persistenceUnitName` property. The `datasource` property is reference to the appropriate `dataSource` bean. The `jpaVendorAdapter` property is set to the `org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter`, that is an implementation for Hibernate `EntityManager`.

Finally, the `transactionManager` bean is defined, using the `org.springframework.transaction.jta.JtaTransactionManager`. It holds two properties to configure. The `transactionManager`, and the `atomikosTransactionManager`. They are references to two beans of `com.atomikos.icatch.jta.UserTransactionManager` class and `com.atomikos.icatch.jta.J2eeUserTransaction` class respectively.

[applicationContext.xml](#)

```

01 <beans xmlns="http://www.springframework.org/schema/beans"
02     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03     xmlns:p="http://www.springframework.org/schema/p"
04     xmlns:aop="http://www.springframework.org/schema/aop"
05     xmlns:context="http://www.springframework.org/schema/context"
06     xmlns:jee="http://www.springframework.org/schema/jee"
07     xmlns:tx="http://www.springframework.org/schema/tx"
08     xmlns:task="http://www.springframework.org/schema/task"
09     xsi:schemaLocation="http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
10     http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
11     http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.2.xsd
12     http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
13     http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
14     http://www.springframework.org/schema/task http://www.springframework.org/schema/task/spring-task-3.2.xsd">
15
16     <context:component-scan base-package="com.javacodegeeks.snippets.enterprise.*" />
17     <tx:annotation-driven />
18     <tx:jta-transaction-manager />
19
20     <bean id="dataSourceA" class="com.atomikos.jdbc.AtomikosDataSourceBean" init-method="init"
21         destroy-method="close">
22         <property name="uniqueResourceName"><value>DataSourceA</value></property>
23         <property name="xaDataSourceClassName">
24             <value>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</value></property>
25         <property name="xaProperties">
26             <props>
27                 <prop key="databaseName">companyA</prop>
28                 <prop key="serverName">localhost</prop>
29                 <prop key="port">3306</prop>
30                 <prop key="user">root</prop>
31                 <prop key="password">root</prop>
32                 <prop key="url">jdbc:mysql://localhost:3306/companyA</prop>
33             </props>
34         </property>
35         <property name="minPoolSize"><value>1</value></property>
36     </bean>
37
38     <bean id="dataSourceB" class="com.atomikos.jdbc.AtomikosDataSourceBean" init-method="init" destroy-
39         method="close">
40         <property name="uniqueResourceName"><value>DataSourceB</value></property>
41         <property name="xaDataSourceClassName">
42             <value>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</value></property>
43         <property name="xaProperties">
44             <props>
45                 <prop key="databaseName">companyB</prop>
46                 <prop key="serverName">localhost</prop>
47                 <prop key="port">3306</prop>
48                 <prop key="user">root</prop>
49                 <prop key="password">root</prop>
50                 <prop key="url">jdbc:mysql://localhost:3306/companyB</prop>
51             </props>
52         </property>
53     </bean>
54
55     <bean id="entityManagerFactoryA" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
56         <property name="persistenceUnitName">companyA</property>
57         <property name="persistenceXmlLocation">persistence.xml</property>
58         <property name="dataSource">dataSourceA</property>
59         <property name="jpaVendorAdapter">jpaVendorAdapter</property>
60     </bean>
61
62     <bean id="entityManagerFactoryB" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
63         <property name="persistenceUnitName">companyB</property>
64         <property name="persistenceXmlLocation">persistence.xml</property>
65         <property name="dataSource">dataSourceB</property>
66         <property name="jpaVendorAdapter">jpaVendorAdapter</property>
67     </bean>
68
69     <bean id="transactionManager" class="org.springframework.transaction.jta.JtaTransactionManager">
70         <property name="transactionManager">transactionManager</property>
71         <property name="atomikosTransactionManager">atomikosTransactionManager</property>
72     </bean>
73
74 </beans>

```

```

39     </props>
40 </property>
41 <property name="minPoolSize"><value>1</value></property>
42 </bean>
43
44 <bean id="entityManagerFactoryA"
45 class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
46   <property name="persistenceXmlLocation">
47     <value>classpath*:persistence.xml</value>
48   </property>
49   <property name="persistenceUnitName" value="PersistenceUnitA" />
50   <property name="dataSource" ref="dataSourceA" />
51   <property name="jpaVendorAdapter">
52     <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
53       <property name="showSql" value="true" />
54       <property name="databasePlatform" value="org.hibernate.dialect.MySQL5InnoDBDialect"
55     />
56   </bean>
57   </property>
58 </bean>
59
60 <bean id="entityManagerFactoryB"
61 class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
62   <property name="persistenceXmlLocation">
63     <value>classpath*:persistence.xml</value>
64   </property>
65   <property name="persistenceUnitName" value="PersistenceUnitB" />
66   <property name="dataSource" ref="dataSourceB" />
67   <property name="jpaVendorAdapter">
68     <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
69       <property name="showSql" value="true" />
70       <property name="databasePlatform" value="org.hibernate.dialect.MySQL5InnoDBDialect"
71     />
72   </bean>
73   </property>
74 </bean>
75
76 <bean id="atomikosTransactionManager" class="com.atomikos.icatch.jta.UserTransactionManager"
77 init-method="init" destroy-method="close">
78   <property name="forceShutdown" value="false" />
79 </bean>
80
81 <bean id="atomikosUserTransaction" class="com.atomikos.icatch.jta.J2eeUserTransaction">
82   <property name="transactionTimeout" value="300" />
83 </bean>
84
85 <bean id="transactionManager" class="org.springframework.transaction.jta.JtaTransactionManager"
86 depends-on="atomikosTransactionManager,atomikosUserTransaction">
87   <property name="transactionManager" ref="atomikosTransactionManager" />
88   <property name="userTransaction" ref="atomikosUserTransaction" />
89   <property name="allowCustomIsolationLevels" value="true" />
90 </bean>
91 </beans>

```

8.3 Configure Web Application Deployment Descriptor

The `web.xml` file is the file that defines everything about your application that a server needs to know. Servlets and other components like filters or listeners, initialization parameters, container-managed security constraints, resources, welcome pages, etc are set here.

The `servlet` element declares the `AppServlet`, and the

`org.springframework.web.context.support.HttpServletRequestHandlerServlet` class that implements it. The `servlet-mapping` element specifies the `/appServlet` URL pattern that invokes the servlet in a browser. In the `context-param` element we set the `contextConfigLocation` parameter, where the `applicationContext.xml` file location is defined. In `listener` element the Bootstrap listener is set to start up Spring's `applicationContext.xml`. The `resource-ref` element is set in both datasources to define a reference lookup name to the resources. This allows the servlet code to look up the resources by a "virtual" name that is mapped to the actual location at deployment time.

`web.xml`

```

01 <?xml version="1.0" encoding="UTF-8"?>
02 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03   xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-
04   app_2_5.xsd"
05   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
06   app_3_0.xsd"
07   id="WebApp_ID" version="3.0">
08
09   <display-name>javacodegeeks</display-name>
10
11   <context-param>
12     <param-name>contextConfigLocation</param-name>
13     <param-value>/WEB-INF/applicationContext.xml</param-value>
14   </context-param>
15
16   <listener>
17     <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
18   </listener>
19
20   <servlet>
21     <display-name>AppServlet</display-name>
22     <servlet-name>appServlet</servlet-name>
23     <servlet-class>org.springframework.web.context.support.HttpServletRequestHandlerServlet</servlet-
24     class>
25   </servlet>

```

```

26 <servlet-mapping>
27   <servlet-name>appServlet</servlet-name>
28   <url-pattern>/appServlet</url-pattern>
29 </servlet-mapping>
30
31 <resource-ref>
32   <description>MySQL DS</description>
33   <res-ref-name>jdbc/DataSourceA</res-ref-name>
34   <res-type>javax.sql.DataSource</res-type>
35   <res-auth>Container</res-auth>
36 </resource-ref>
37
38 <resource-ref>
39   <description>MySQL DS</description>
40   <res-ref-name>jdbc/DataSourceB</res-ref-name>
41   <res-type>javax.sql.DataSource</res-type>
42   <res-auth>Container</res-auth>
43 </resource-ref>
44
45 </web-app>

```

9. Run the application in Tomcat

In order to run the application in tomcat we first have to build the project. The war produced is placed at webapps folder of tomcat. Then, we startup the server. After hitting on

localhost:8080/springexample/appServlet

in a browser, we can check on MySQL, that in both databases, companyA and companyB the tables EmployeeA and EmployeeB have one record. The message returned in the browser is the one below:



10. Rollback case

Now, let's see what happens if one of the two transactions fail. We will change the persistEmployee(EmployeeB employee) method of EmployeeBDAOImpl.java class so as to throw an Exception.

EmployeeBDAOImpl.java

```

01 package com.javacodegeeks.snippets.enterprise.dao;
02
03 import javax.persistence.EntityManager;
04 import javax.persistence.PersistenceContext;
05
06 import org.springframework.stereotype.Service;
07
08 import com.javacodegeeks.snippets.enterprise.model.EmployeeB;
09
10 @Service
11 public class EmployeeBDAOImpl implements EmployeeBDAO {
12
13     @PersistenceContext(unitName="PersistenceUnitB")
14     private EntityManager entityManager;
15
16     public void persistEmployee(EmployeeB employee) throws Exception {
17         // entityManager.persist(employee);
18         throw new Exception();
19     }
20 }

```

We build the project again and place the new war file in webapps file of tomcat. After starting up tomcat again, the result is the one below:



This is caused because since one of the transactions throws an exception the distributed transaction rolls back too.

This was an example of JTA multiple resource transactions in a Tomcat server, using Atomikos Transaction Manager.

Download the Eclipse project of this tutorial : [SpringJTAatomikosTomcatExample.zip](#)

You might also like:

- [Spring 3.1 profiles and Tomcat configuration](#)
- [Spring Declarative Transactions Example](#)
- [Spring Transactions Visibility](#)

Related Whitepaper:

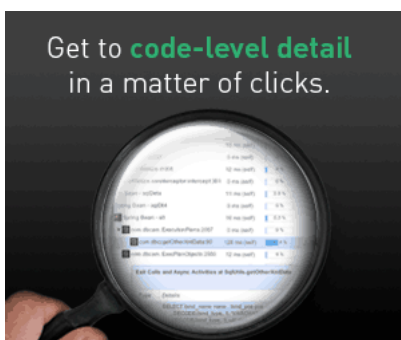


Java EE 6 Cookbook for Securing, Tuning, and Extending Enterprise Applications

Java Platform, Enterprise Edition is a widely used platform for enterprise server programming in the Java programming language.

This book covers exciting recipes on securing, tuning and extending enterprise applications using a Java EE 6 implementation. The book starts with the essential changes in Java EE 6. Then they will dive into the implementation of some of the new features of the JPA 2.0 specification, and look at implementing auditing for relational data stores. They will then look into how they can enable security for their software system using Java EE built-in features as well as using the well-known Spring Security framework. They will then look at recipes on testing various Java EE technologies including JPA, EJB, JSF, and Web services. Next they will explore various ways to extend a Java EE environment with the use of additional dynamic languages as well as frameworks. At the end of the book, they will cover managing enterprise application deployment and configuration, and recipes that will help you debug problems and enhance the performance of your applications.


[Get it Now!](#)



Share and enjoy!


2 Responses to "Spring JTA multiple resource transactions in Tomcat with Atomikos example"

Somen

- 

December 5th, 2013 at 10:30 am

Nice tutorial. Thanks!

[Reply](#)
- 

Lalit Jha

December 15th, 2013 at 4:44 pm

Awesome tutorial.....Thanks for demonstrating using latest version of Spring

[Reply](#)

Leave a Reply

Name (Required)

Mail (will not be published) (Required)

Website

– two = 2

☐ Notify me of followup comments via e-mail

☒ Sign me up for the newsletter!

Submit Comment

Knowledge Base	Partners	Hall Of Fame	About Java Code Geeks
Examples	Mkyong	“Android Full Application Tutorial” series	<p>JCGs (Java Code Geeks) is an independent online community for creating the ultimate Java to Java developers resource center; the technical architect, technical team lead (senior developer), project junior developers alike. JCGs serve the Java, SOA, Agile and TDD communities with daily news written by domain experts, articles, 1 announcements, code snippets and open source projects.</p>
Resources	The Code Geeks Network	GWT 2 Spring 3 JPA 2 Hibernate 3.5 Tutorial	
Tutorials		Android Game Development Tutorials	
Whitepapers		Android Google Maps Tutorial	
		.NET Code Geeks	
	Web Code Geeks	Funny Source Code Comments	
		Java Best Practices – Vector vs ArrayList vs HashSet	
		Android JSON Parsing with Gson Tutorial	
		Android Quick Preferences Tutorial	