



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 Submitting

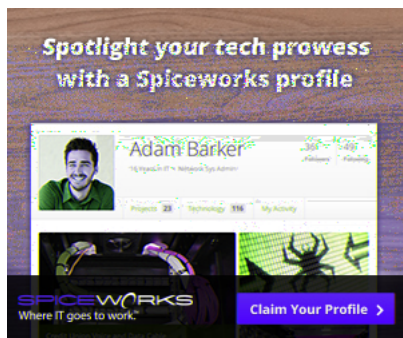
Java Android JVM Languages Software Development Agile DevOps Communications Career Misc Meta JCG

Search this site...

<p>Co e Ja a</p> <p>En e p i e Ja a</p> <p>De k o p Ja a</p> <p>With a la ge n mbe of p ojec angling f o m p o g amming and of a e enginee ing o elecomm nica ion , ha d a e de ign and anal i .</p> <p></p>	<p>Ja a Spring JTA m l i p l e e o c e an ac ion in Tomca i h A omiko e ample</p> <p>Feb</p> <p>Lora Frangkouli</p> <p>g ad a ed f o m Comp e Enginee ing and Info ma ic Depa men in he Uni e i of Pa a . She al o hold a in Economic f o m he Na iona and Technical Uni e i of A hen . D ing he die he ha been in ol ed</p>	<p>Advertise Here</p> <p>Advertise Here</p>
--	---	---

Spring JTA multiple resource transactions in Tomcat with Atomikos example

by Theodor Fagkion on July 3, 2013 Filed in: Enterprise Java Tags: Apache Tomcat, Atomikos, JTA, Spring



rollback .

Open the development environment in [Eclipse](#). We are using Eclipse JNo (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_ _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.

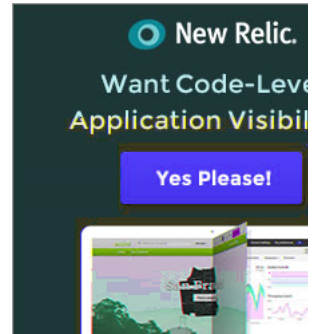
Let's begin,

1. Create a new Maven project

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

In this tutorial we will see how to implement JTA multiple resource transaction in a Tomcat server, using [Atomikos Transaction Manager](#). Atomikos transaction manager provides support for distributed transaction. The distributed transaction, often called global transaction, has multiple committed in a coordinated manner. The distributed transaction is described by the XA standard. XA goes on how a transaction manager (such as Atomikos) can tell a database how to go on a part of the transaction, and how to coordinate the two-phase commit (2PC) protocol at the end of each transaction.

Here, we will create simple Entity class mapped to two different database and we will create object of the class on the database using one distributed transaction. We will also see what happens when one of the participating transaction



Newsletter



Join our Newsletter
e cl i e acc
ne in he J
ell a in gh
And oid, Sca
o he ela ed
A an extra l
joining o

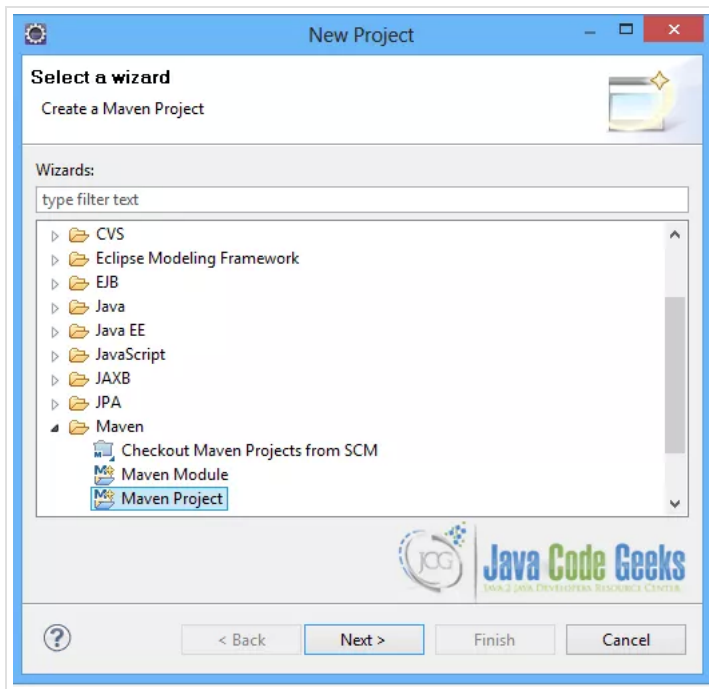
new e-books, published by Java Code Geeks
Join JCG partner for reading

Join Us

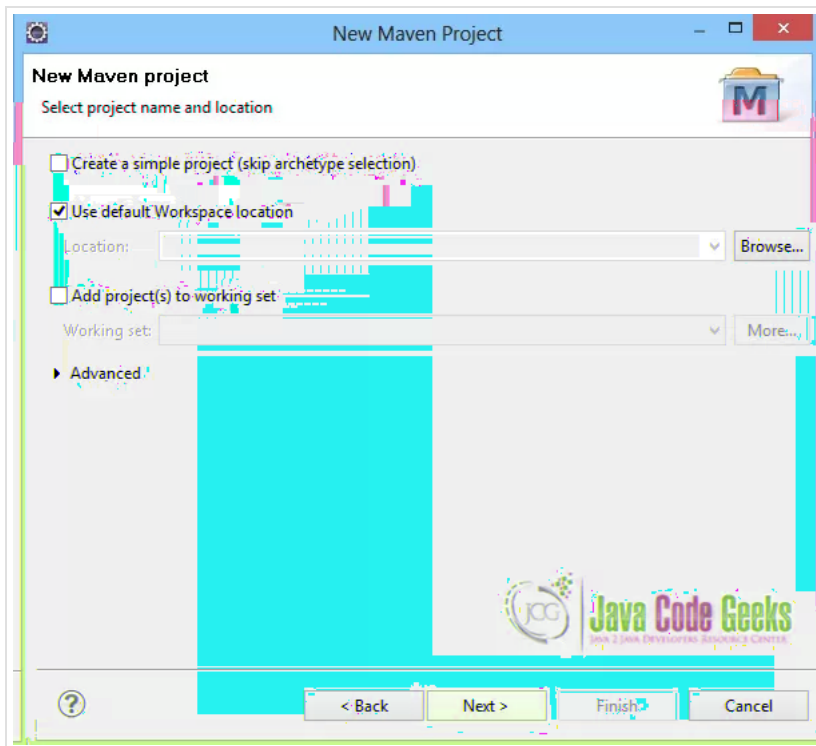


With 469,791
i io and c
a ho ea
he op Ja a
a o nd. Con
he looko fr
enco age c

If you have a blog in Hindi and you want to monetize it, you should check out our partner program. You can also be a part of the Java Code Geeks community and help us grow. We are looking for more people to join our team and help us create more content. If you are interested, please contact us at [info@java-code-geeks.com](#).



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



Here the main archetype for creating a web application must be added. Click on **"Add Archetype"** and add the archetype. Set the ArchetypeGroupId attribute to "org.apache.maven.archetypes", the ArchetypeArtifactId attribute to "maven-archetype-webapp" and the ArchetypeVersion to "1.0". Click on **"OK"** to continue.

Carrer Opportunities

Scien i -Cogni i e S em (FUL
Feb a 5 h, 2014

Sof a e De elope , 2 (FULL-TIME
4 h, 2014

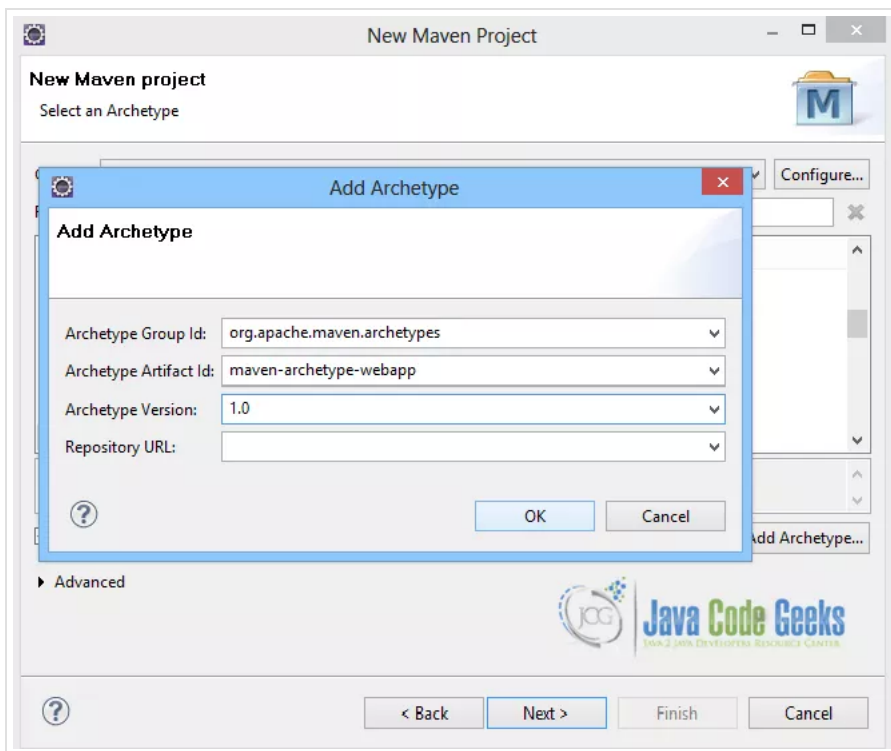
J nio Ja a Enginee (FULL-TIME
2014

Sof a e De elopmen Enginee ir
TIME) Feb a 4 h, 2014

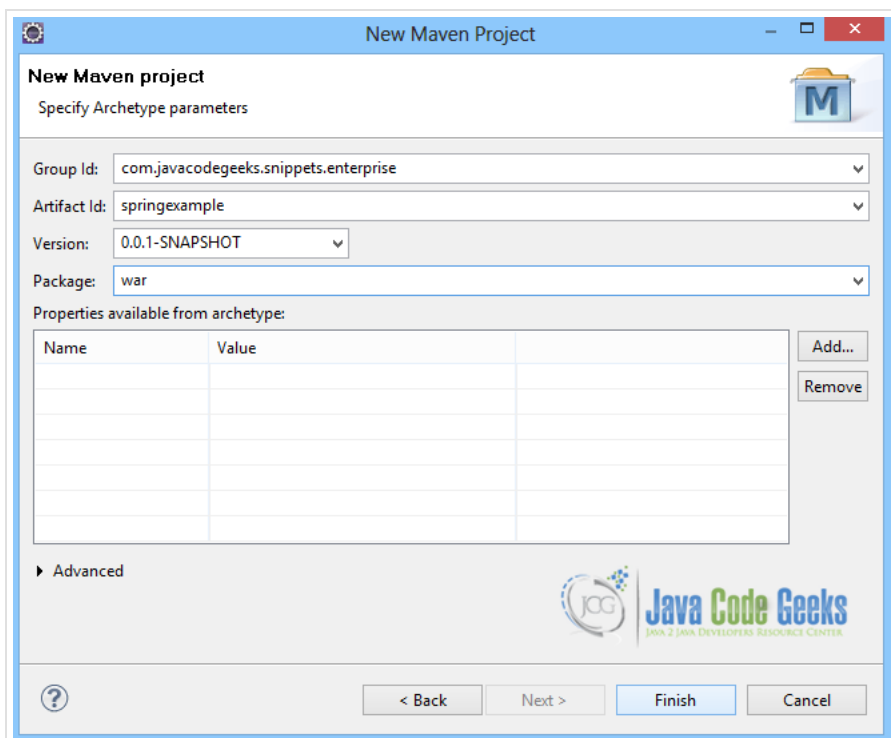
En e p i e Ja e Sof a e De elop
TIME) Feb a 4 h, 2014

Tags

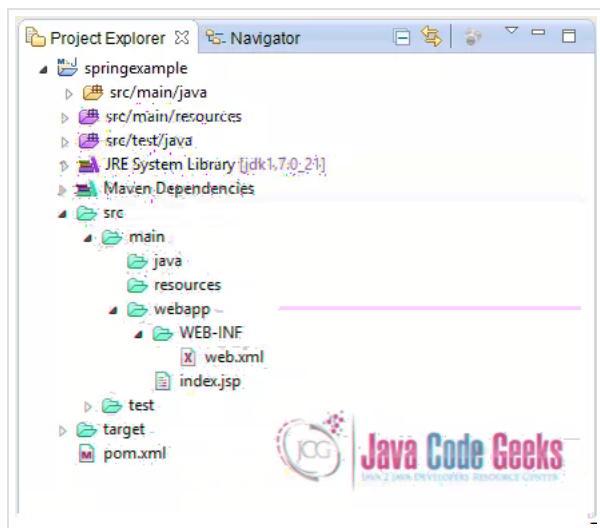
Akka And old T o ial Apache Camel
Apache Ma en Apache Tom
Clo d Conc enc
De ign Pa e n Eclip e ci
G ail IDE In e ie Ja a 7 Ja z
Ja aFX Ja aOne JAXB
JBo Hibe na e JPA JSF
JVM Logging MongoDB
O acle Gla Fi h Pe fo mance
P ojec Managemen RESTf I W
Sec i Sp ing s
Sp ing MVC Sp ing Sec i T



In the Enter an artifact id page of the wizard, you can define the name and main package of the project. Set the Group Id attribute to "com.javacodegeeks.snippets.enterprise" and the Artifact Id attribute to "springexample". The aforementioned selection composes the main project package as "com.javacodegeeks.snippets.enterprise.springexample" and the project name as "springexample". Set the Package attribute to "war", so that a file will be created to be deployed to Tomcat easily. Hit Finish on the wizard and create the project.



The Maven project creation is shown below:



I contain of the following folders :

- / src/main/java folder, contains source files for the application,
- / src/main/resources folder contains all resource files,
- / src/main/resources folder contains configuration file,
- / target folder contains the compiled and packaged deliverable,
- / src/main/resources/webapp/WEB-INF folder contains the deployment descriptor for the Web application,
- the pom.xml is the project object model (POM) file. The single file 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 change :
Change the property value of **org.springframework.version** and set it to **3.2.3.RELEASE**.
- Navigate to the Dependencies page of the POM editor and create the following dependency (click on the Add button in the top right corner of the Dependencies page):
GroupId : **org.springframework** ArtifactId : **spring-web** Version : **\${org.springframework.version}**

Alternatively, you can add the Spring dependency in Maven `pom.xml` file, by editing 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>

```

Alternatively, you can manage library dependencies declaratively. A local repository is created (by default under `~/.m2` folder) and all required libraries are downloaded and placed in the local repository. From there, you can manage library dependencies declaratively.

3. Add all required dependencies

All dependencies needed to perform action management are as follows.

`pom.xml`:

```

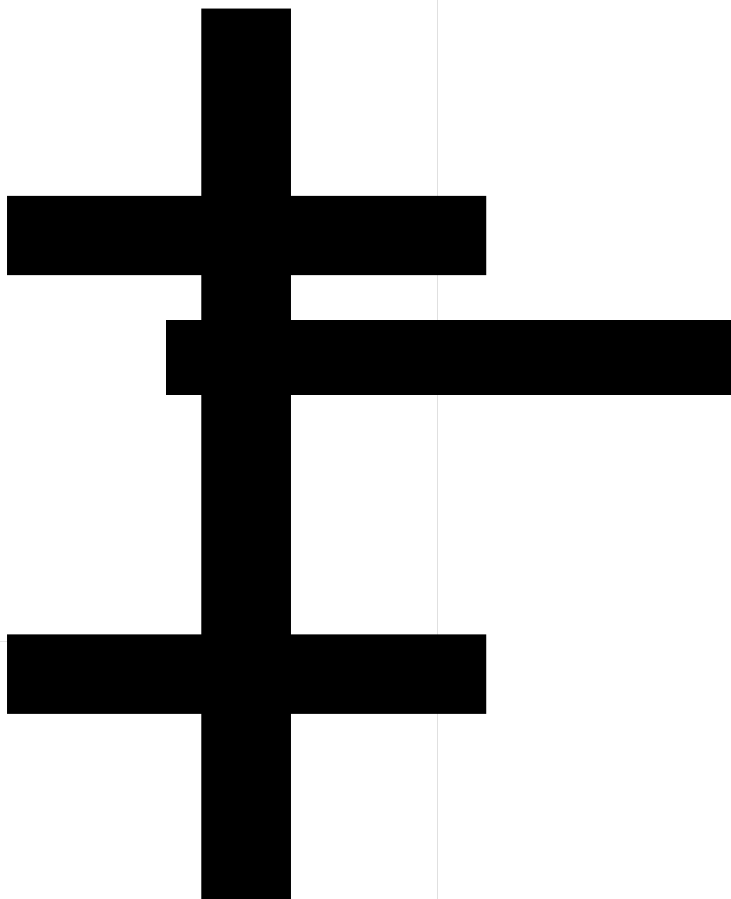
001 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
002 instance"
003 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

```

```

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</

```



```

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. The javax.persistence annotations to be mapped to database, EMPLOYEEA and EMPLOYEEB in different databases. In particular, the @Entity annotations specify that each class is an entity. The @Table annotations specify the primary table for the annotated entity. The @Column annotations are used to specify a mapped column for the entity field, 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 Database Access Objects implemented are the `EmployeeADAOImpl.java` and `EmployeeBDAOImpl.java` classes. They are annotated with the `@Service` annotation, declaring that they are Spring Beans and handling Spring's auto-declaration. They both use the `javax.persistence.EntityManager` object in each of the database.

An `EntityManager` instance is associated with a persistence context. A persistence context is a set of entities in an instance in which for an entity is identified by its identifier in the instance. Within the persistence context, the entity instance and its lifecycle are managed. The `EntityManager` API is used to create and remove entities in the instance, to find entities by their primary key, and to query the entities. The `EntityManager` is configured in `persistence.xml` file, has its description in paragraph 8.1.

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

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

Basically, the methods implemented in both DAOs, using the `persist(Object entity)` API method of `EntityManager` to create an object in 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 }

```

EmployeeADAOImpl.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 | }
```

EmployeeBDAOImpl.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. Then, in the `persistEmployees(EmployeeA employeeA, EmployeeB employeeB)` method implemented here, the DAO methods are invoked to perform the database action in the database. The `EmployeeServiceImpl.java` class is also annotated with the `@Service` annotation, declaring it as a Spring Bean and handling Spring's auto-declaration.

The `@Transactional` annotation is placed before the method, indicating that the action is executed when the method is invoked. The action is a global constraint managed action 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 implements the `org.springframework.web.HttpRequestHandler` and overrides the `handleRequest(HttpServletRequest req, HttpServletResponse resp)` API method. The `EmployeeService` is injected here, i.e., the `@Autowired` annotation. It is used in the `handleRequest(HttpServletRequest req, HttpServletResponse resp)` API method to perform a new `EmployeeA` and a new `EmployeeB` object. The method also contains a check message if the method execution fails and rollback message if the method has 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.RequestHandler;
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 RequestHandler {
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 are associated with the database configuration in `persistence.xml` file. Here we define the persistence unit element, and define the entity class associated with 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 let the package have contain all classes that have the `@Component` annotation in the Spring bean.

The `<tx:annotation-driven/>` element allows to have Spring inject `@Transactional` annotation on configuration of the application beans in the transactional behavior.

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

In `dataSourceA` and `dataSourceB` beans we define the data source. The `com.atomikos.jdbc.AtomikosDataSourceBean` is the class we use for Atomikos JTA-enabled connection pooling. It has two properties to configure. The `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource` class is the `xaDataSourceClass` property, the value in the `xaProperties` element can be the property (name, value) to configure the `XADataSource`.

In `entityManagerFactoryA` and `entityManagerFactoryB` beans we use the `org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean` class. It is a `FactoryBean` that creates a `JPA EntityManagerFactory` according to JPA standard container bootstrapping. We can use the `persistence.xml` location in the `persistenceXmlLocation` property. We can use the name of the persistence unit declared in the `EntityManagerFactory`, in the `persistenceUnitName` property. The `datasource` property is reference to the application `dataSource` bean. The `jpaVendorAdapter` property is the `org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter`, has 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`. The value reference to the bean 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
10         http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
11         http://www.springframework.org/schema/beans
12         http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
13         http://www.springframework.org/schema/context
14         http://www.springframework.org/schema/context/spring-context-3.2.xsd
15         http://www.springframework.org/schema/jee
16         http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
17         http://www.springframework.org/schema/tx
18         http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
19         http://www.springframework.org/schema/task
20         http://www.springframework.org/schema/task/spring-task-3.2.xsd">
21
22     <context:component-scan base-package="com.javacodegeeks.snippets.enterprise.*" />
23     <tx:annotation-driven />
24     <tx:jta-transaction-manager />
25
26     <bean id="dataSourceA" class="com.atomikos.jdbc.AtomikosDataSourceBean" init-method="init"
27         destroy-method="close">
28         <property name="uniqueResourceName"><value>DataSourceA</value></property>
29         <property name="xaDataSourceClassName">
30             <value>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</value></property>
31         <property name="xaProperties">
32             <props>
33                 <prop key="databaseName">companyA</prop>
34                 <prop key="serverName">localhost</prop>
35                 <prop key="port">3306</prop>
36                 <prop key="user">root</prop>
37                 <prop key="password">root</prop>
38                 <prop key="url">jdbc:mysql://localhost:3306/companyA</prop>
39             </props>
40         </property>
41         <property name="minPoolSize"><value>1</value></property>
42     </bean>
43
44     <bean id="dataSourceB" class="com.atomikos.jdbc.AtomikosDataSourceBean" init-method="init" destroy-
45         method="close">
46         <property name="uniqueResourceName"><value>DataSourceB</value></property>
47         <property name="xaDataSourceClassName">
48             <value>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</value></property>
49         <property name="xaProperties">
50             <props>
51                 <prop key="databaseName">companyB</prop>
52                 <prop key="serverName">localhost</prop>
53                 <prop key="port">3306</prop>
54                 <prop key="user">root</prop>
55                 <prop key="password">root</prop>
56                 <prop key="url">jdbc:mysql://localhost:3306/companyB</prop>
57             </props>
58         </property>
59     </bean>
60
61     <bean id="transactionManager" class="org.springframework.transaction.jta.JtaTransactionManager">
62         <property name="transactionManager"><ref bean="transactionManager"></ref></property>
63         <property name="atomikosTransactionManager"><ref bean="atomikosTransactionManager"></ref></property>
64     </bean>
65
66     <bean id="entityManagerFactoryA" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
67         <property name="dataSource"><ref bean="dataSourceA"></ref></property>
68         <property name="persistenceUnitName">companyA</property>
69         <property name="persistenceXmlLocation">classpath:/META-INF/persistence.xml</property>
70         <property name="jpaVendorAdapter"><ref bean="jpaVendorAdapter"></ref></property>
71     </bean>
72
73     <bean id="entityManagerFactoryB" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
74         <property name="dataSource"><ref bean="dataSourceB"></ref></property>
75         <property name="persistenceUnitName">companyB</property>
76         <property name="persistenceXmlLocation">classpath:/META-INF/persistence.xml</property>
77         <property name="jpaVendorAdapter"><ref bean="jpaVendorAdapter"></ref></property>
78     </bean>
79
80     <bean id="jpaVendorAdapter" class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
81         <property name="showSql">true</property>
82     </bean>
83
84     <bean id="transactionManager" class="com.atomikos.icatch.jta.UserTransactionManager">
85         <property name="transactionManager"><ref bean="transactionManager"></ref></property>
86     </bean>
87
88     <bean id="atomikosTransactionManager" class="com.atomikos.icatch.jta.J2eeUserTransaction">
89         <property name="transactionManager"><ref bean="transactionManager"></ref></property>
90     </bean>
91
92 </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 the application that we need to know. See the end of the component like file to list the initialization parameters, container-managed entities, context, welcome page, etc. etc. etc.

The servlet element declares the `AppServlet`, and the

`org.springframework.web.context.support.HttpServletRequestHandlerServlet` class implements it. The servlet-mapping element specifies the `/appServlet` URL pattern in the `web.xml` file. In the `context-param` element, the `contextConfigLocation` parameter, the `applicationContext.xml` file location is defined. In the `listener` element, the `Bootstrap` listener is the `Spring` `applicationContext.xml`. The `resource-ref` element is in the `web.xml` file to define the resource lookup name of the `context`. This allows the code to look up the `context` by its actual name has mapped to the actual location of the deployment.

`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>

```




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

This is an example of JTA multiple resource transaction in a Tomcat server, using Atomikos Transaction Manager. Download the Eclipse project of this tutorial: [SpringJTAAtomikos Tomcat Example](#). [ip](#)

You might also like:

- [Spring 3.1 profile and Tomcat configuration](#)
- [Spring Declarative Transaction Example](#)
- [Spring Transaction Visibility](#)

Related Whitepaper:

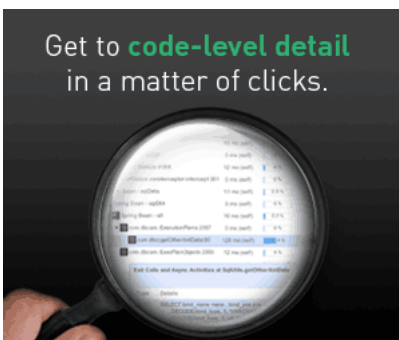


Java EE 6 Cookbook for Securing, Tuning, and Extending Enterprise Applications

Java Platform, Enterprise Edition is a standard platform for enterprise development in the Java programming language.

This book covers creating recipes on securing, tuning and extending enterprise application using a Java EE 6 implementation. The book also includes the essential change in Java EE 6. Then the author will describe how to implement some of the new features of the JPA 2.0 specification, and look at implementing a distributed relational database. The author will then look into how the can enable security for the use of a new emerging Java EE built-in features, allowing the well-known Spring Security framework. The author will then look at recipes on using a Java EE technologies including JPA, EJB, JSF, and Web services. Next the author will explore a database and a Java EE environment in the use of additional dynamic language, allowing a framework. At the end of the book, the author will cover managing enterprise application deployment and configuration, and recipes that will help to debug problems and enhance the performance of an application.


[Get it Now!](#)



Share and enjoy!

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

Somen



Decembe 5 h, 2013 a 10:30 am

Nice o ial. Thank !

[Repl](#)



Lalit Jha
Decembe 15 h, 2013 a 4:44 pm

A e ome o ial ..Thank fo demon a ing ing la e e ion of Sp ing

[Repl](#)

Leave a Reply

Name (Req i ed)

Mail (ill no be p bli hed) (Req i ed)

Web i e

o = 2

☐ No if me of follo p commen ia e-mail

☒ Sign me p fo he ne le e !

S bmi

Commen

Knowledge Base	Partners	Hall Of Fame	About Java Code Geeks
E ample	Mk ong	And oid F ll Applica ion T o ial e ie	JCG (Ja a Code Geek) i an independen online comm ni f c ea ing he l ima e Ja a o Ja a de elope e o ce cen e ; i
Re o ce		GWT 2 Sp ing 3 JPA 2 Hibe na e 3.5 T o ial	echanical a chi ec , echnical eam lead (enio de elope), p oje
T o ial	The Code Geeks Network	And oid Game De elopmen T o ial	j nio de elope alike. JCG e e he Ja a, SOA, Agile and Te
Whi epape	Ja a Code Geek	And oid Google Map T o ial	comm nie i h dail ne i en b domain e pe , a icl e ,
	.NET Code Geek	And oid Loca ion Ba ed Se ice Applica ion GPS loca ion	anno ncemen , code nippe and open o ce p ojec .
	Web Code Geek	F nn So ce Code Commen	
		Ja a Be P ac ice Vec o A a Li Ha hSe	
		And oid JSON Pa ing i h G on T o ial	
		And oid Q ick P efe ence T o ial	