Java SE: Unit Testing CDI with JUnit and JBoss Weld SE

BY MEMORYNOTFOUND · PUBLISHED APRIL 13, 2015 · UPDATED APRIL 4, 2016

In this tutorial we will show you how to use Java EE Context Dependency Injection in a Java SE environment by Unit Testing CDI with JUnit and JBoss Weld. To enable CDI in our JUnit tests we created a custom JUnit runner which will enable us to use Java EE annotations in our Unit tests. CDI is the Java standard for dependency injection and part of the EE specification. Weld is a reference implementation of CDI developed by JBoss.

Project structure

```
src
|--main
|    +--java
|        +--com
|            +--memorynotfound
|                +--cdi
|                    |--CourseService.java
|                    |--CourseServiceImpl.java
|                    |--Log.java
|                    |--LoggingProducer.java
|    +--resources
|        +--META-INF
|            |--beans.xml
|        |--logback.xml
|--test
|    +--java
|        +--com
|            +--memorynotfound
|                +--cdi
|                    +--test
|                        |--LogTest.java
|                        |--WeldContainer.java
|                        |--WeldJUnit4Runner.java
|    +--resources
|        +--META-INF
|            |--beans.xml
pom.xml
```

Maven Dependencies

For this tutorial we'll require the following dependencies. `javaee-api` for the java ee CDI dependencies. `org.jboss.weld` as the reference implementation of CDI and we will use `junit` for our unit testing.

```
<!-- api -->
<dependency>
    <groupId>javax</groupId>
    <artifactId>javaee-api</artifactId>
    <version>7.0</version>
    <scope>provided</scope>
</dependency>
<!-- logging -->
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.1.2</version>
</dependency>
<!-- testing -->
<dependency>
    <groupId>org.jboss.weld.se</groupId>
```

```
    <artifactId>weld-se</artifactId>
    <version>2.2.10.Final</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
</dependency>
```

Creating a producer using the `@Produces` annotation

In a Java EE environment it is common to create loggers for every class. The logging producer will enable us to inject a `org.slf4j.Logger` implementation into any class we want without having to configure them every time. We create the producer by annotation our method using the `@Produces` method. This annotation lets us inject the Logger implementation into any CDI bean. To configure our logger we need to obtain the class of where our logger will be injected. We can get this information from the `InjectionPoint` e.g.: `injectionPoint.getMember().getDeclaringClass()`.

```
package com.memorynotfound.cdi;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import javax.enterprise.inject.Produces;
import javax.enterprise.inject.spi.InjectionPoint;

public class LoggingProducer {

    @Produces @Log
    private Logger createLogger(InjectionPoint injectionPoint) {
        return LoggerFactory.getLogger(injectionPoint.getMember().getDec
    }

}
```

Creating the qualifier using the `@Qualifier` annotation

We created a `@Log Qualifier` describes the injection and enables a type safe way to inject an injection into an injection point.

```
package com.memorynotfound.cdi;

import javax.inject.Qualifier;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

import static java.lang.annotation.ElementType.*;
import static java.lang.annotation.RetentionPolicy.RUNTIME;

@Qualifier
@Retention(RUNTIME)
@Target({FIELD, TYPE, METHOD})
public @interface Log {
```

```
}
```

Lets start by building an interface

```
package com.memorynotfound.cdi;

public interface CourseService {

    void registerCourse(String course);

}
```

And the concrete implementation

Here we inject a `org.slf4j.Logger` using the `@Inject` annotation the producer produces from our `LoggingProducer` is described with the `@Log` Qualifier which tells CDI to inject a `Logger` which has the `@Log` Qualifier specified.

```
package com.memorynotfound.cdi;

import org.slf4j.Logger;
import javax.inject.Inject;

public class CourseServiceImpl implements CourseService {

    @Inject @Log
    private Logger LOG;

    @Override
    public void registerCourse(String course) {
        LOG.info("adding course: " + course);
    }

}
```

Enabling CDI using beans.xml file

Remember when working with CDI you must include a `beans.xml` file into every jar/war/ear file where you want to use CDI. This file can be completely empty. Just make sure this file is under the META-INF or WEB-INF folder on your classpath.

```
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                           http://xmlns.jcp.org/xml/ns/javaee/beans_1_1.

</beans>
```

Configuring our logback logger

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{yyyy-MM-dd HH:mm:ss} | %-5p | [%thread] %logger{
        </encoder>
    </appender>

    <logger name="com.memorynotfound" level="TRACE"/>

    <root level="WARN">
        <appender-ref ref="STDOUT" />
    </root>

</configuration>
```

Unit Testing CDI with JUnit in a Java SE environment

This isn't actually a real Unit test because it is not making any asserts. I just created this test to show you how you can leverage CDI in a Java SE environment while unit testing. We can use the `@Inject` annotation in our test cases because we enable this using our custom `WeldJUnit4Runner` .

```java
package com.memorynotfound.cdi.test;

import com.memorynotfound.cdi.CourseService;
import org.junit.Test;
import org.junit.runner.RunWith;
import javax.inject.Inject;

@RunWith(WeldJUnit4Runner.class)
public class LogTest {

    @Inject
    private CourseService courseService;

    @Test
    public void testCDI() {
        courseService.registerCourse("Unit Testing CDI in a Java SE envi
    }

}
```

Creating a WeldContext Utility class

Before creating the `WeldJUnit4Runner` we created a simple utility class that will instantiate the Weld Container which enables the CDI for our Java SE application. We create an additional `addShutdownHook` which will be called when the pre-destroy phase is called. With CDI you cannot create an Object using the `new` keyword otherwise the lifecycle of that bean will not be managed by the container. That's why we created a helper method `getBean()` that'll get an instance of a bean.

```java
package com.memorynotfound.cdi.test;

import org.jboss.weld.environment.se.Weld;
import org.jboss.weld.environment.se.WeldContainer;
```

```
public class WeldContext {

    public static final WeldContext INSTANCE = new WeldContext();

    private final Weld weld;
    private final WeldContainer container;

    private WeldContext() {
        this.weld = new Weld();
        this.container = weld.initialize();
        Runtime.getRuntime().addShutdownHook(new Thread() {
            @Override
            public void run() {
                weld.shutdown();
            }
        });
    }

    public <T> T getBean(Class<T> type) {
        return container.instance().select(type).get();
    }
}
```

WeldJUnit4Runner

The `WeldJUnit4Runner` extends from `BlockJUnit4ClassRunner` which overrides the `createTest` method. With this method we instantiate all the beans of our test classes. In order to be eligible for CDI.

```
package com.memorynotfound.cdi.test;

import org.junit.runners.BlockJUnit4ClassRunner;
import org.junit.runners.model.InitializationError;

public class WeldJUnit4Runner extends BlockJUnit4ClassRunner {

    public WeldJUnit4Runner(Class<Object> clazz) throws InitializationEr
        super(clazz);
    }

    @Override
    protected Object createTest() {
        final Class<?> test = getTestClass().getJavaClass();
        return WeldContext.INSTANCE.getBean(test);
    }
}
```

```
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                           http://xmlns.jcp.org/xml/ns/javaee/beans_1_1.

</beans>
```

Running mvn test

Just a simple visual verification that all the CDI Dependency Injection is working, The logger gets injected into the `CourseServiceImpl` and we are able to print an info log to our console.

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------
```

```
[INFO] Building Java EE - producers 1.0.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ pr
[INFO] Copying 2 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ produce
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResourc
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ producers
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.memorynotfound.cdi.test.LogTest
2015-04-13 19:04:23 | INFO  | [main] c.m.c.CourseServiceImpl:13 - adding
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.848 se

Results :

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

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.098 s
[INFO] Finished at: 2015-04-13T19:04:23+02:00
[INFO] Final Memory: 9M/245M
[INFO] ------------------------------------------------------------------------
```

References

- Java EE Specification
- JBoss Weld Documentation
- BlockJUnit4ClassRunner JavaDoc