# @Asynchronous @PostConstruct

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

Placing `@Asynchronous` on the `@PostConstruct` of an EJB is not a supported part of Java EE, but this example shows a pattern which works just as well with little effort.

The heart of this pattern is to:

- pass the construction "logic" to an `@Asynchronous` method via a `java.util.concurrent.Callable`
- ensure the bean does not process invocations till construction is complete via an `@AroundInvoke` method on the bean and the `java.util.concurrent.Future`

Simple and effective.  The result is a faster starting application that is still thread-safe.

```java
package org.superbiz.asyncpost;

import javax.annotation.PostConstruct;
import javax.ejb.EJB;
import javax.ejb.Lock;
import javax.ejb.LockType;
import javax.ejb.Singleton;
import javax.interceptor.AroundInvoke;
import javax.interceptor.InvocationContext;
import java.util.concurrent.Callable;
import java.util.concurrent.Future;

import static java.util.concurrent.TimeUnit.SECONDS;

@Singleton
@Lock(LockType.READ)
public class SlowStarter {

    @EJB
    private Executor executor;

    private Future construct;

    private String color;
    private String shape;

    @PostConstruct
    private void construct() throws Exception {
        construct = executor.submit(new Callable() {
            @Override
            public Object call() throws Exception {
                Thread.sleep(SECONDS.toMillis(10));
                SlowStarter.this.color = "orange";
                SlowStarter.this.shape = "circle";
                return null;
            }
        });
    }

    @AroundInvoke
    private Object guaranteeConstructionComplete(InvocationContext context) throws
Exception {
        construct.get();
        return context.proceed();
    }

    public String getColor() {
        return color;
    }

    public String getShape() {
        return shape;
    }
}
```

The `Executor` is a simple pattern, useful for many things, which exposes an interface functionaly equivalent to `java.util.concurrent.ExecutorService`, but with the underlying thread pool controlled by the container.

```
package org.superbiz.asyncpost;

import javax.ejb.AsyncResult;
import javax.ejb.Asynchronous;
import javax.ejb.Lock;
import javax.ejb.LockType;
import javax.ejb.Singleton;
import java.util.concurrent.Callable;
import java.util.concurrent.Future;

@Singleton
@Lock(LockType.READ)
public class Executor {

    @Asynchronous
    public <T> Future<T> submit(Callable<T> task) throws Exception {
        return new AsyncResult<T>(task.call());
    }

}
```

Finally a test case shows the usefulness of `@AroundInvoke` call in our bean that calls `construct.get()`

```java
package org.superbiz.asyncpost;

import junit.framework.Assert;
import org.junit.Test;

import javax.ejb.EJB;
import javax.ejb.embeddable.EJBContainer;

public class SlowStarterTest {

    @EJB
    private SlowStarter slowStarter;

    @Test
    public void test() throws Exception {

        // Start the Container
        EJBContainer.createEJBContainer().getContext().bind("inject", this);

        // Immediately access the fields initialized in the PostConstruct
        // This will fail without the @AroundInvoke call to construct.get()
        Assert.assertEquals("orange", slowStarter.getColor());
        Assert.assertEquals("circle", slowStarter.getShape());
    }
}
```