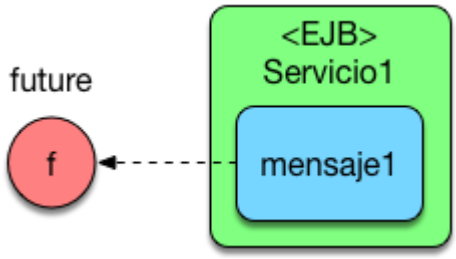
El concepto de Java Futures esta ligado a la programación asíncrona . Vamos a usar Java Futures con Enterprise Java Beans para entender su utilidad. Para ello partiremos de dos 2 EJBs asíncronos que ejecutaran una tarea que tarda 5s y devuelven un Java Future.



¿Para que sirven los Java Futures?

Un Java Future es un objeto que se construye para albergar un valor en un futuro. Vamos a verlos funcionando en código.

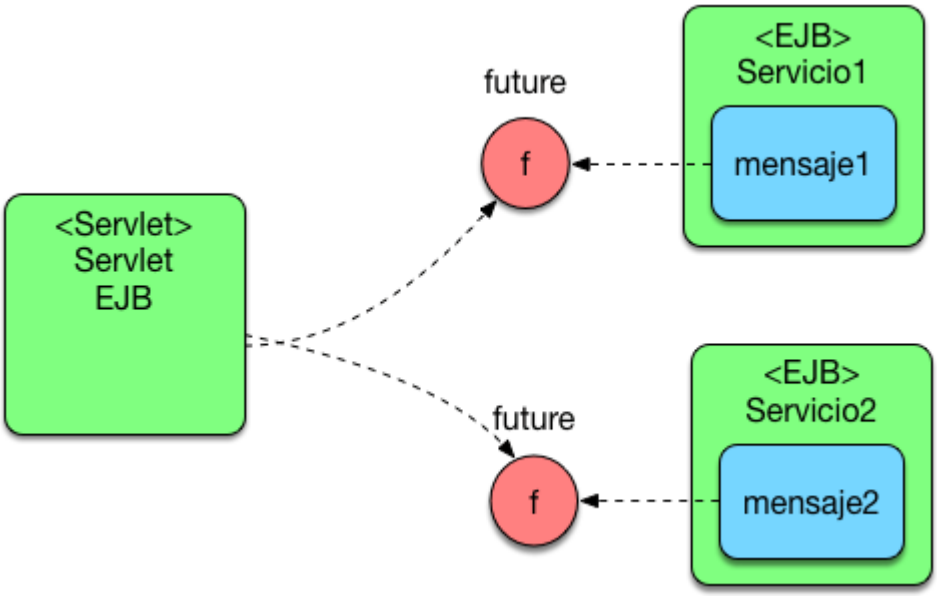
```
package com.arquitecturajava.ejb;

import java.util.concurrent.Future;

import javax.ejb.AsyncResult;
```

```
import javax.ejb.Asynchronous;
import javax.ejb.LocalBean;
import javax.ejb.Stateless;

@Stateless
@LocalBean
public class ServicioEJB2 {

 @Asynchronous
 public Future<String> getMensaje2() {
 try {
 Thread.sleep(5000);
 } catch (InterruptedException e) {
 // TODO Auto-generated catch block
 e.printStackTrace();
 }

 return new AsyncResult<String>("mensaje2 lento");
 }

}
```

```
package com.arquitecturajava.ejb;

import java.util.concurrent.Future;

import javax.ejb.AsyncResult;
import javax.ejb.Asynchronous;
import javax.ejb.LocalBean;
```

```
import javax.ejb.Stateless;

@Stateless
@LocalBean
public class ServicioEJB1 {


 @Asynchronous
 public Future<String> getMensaje1() {
 try {
 Thread.sleep(5000);
 } catch (InterruptedException e) {
 // TODO Auto-generated catch block
 e.printStackTrace();
 }

 return new AsyncResult<String>("mensaje1 lento");
 }

}
```

En este caso tenemos dos EJBs y cada uno tiene un método getMensaje() . Con una programación normal cada EJB devolvería un String . Sin embargo en este caso al tratarse de EJBS asíncronos cada uno devuelve un Java Future a través de una variable de tipo AsyncResult. Vamos a usar un Servlet para acceder a ellos.



El código del servlet:

```
package com.arquitecturajava.servlets;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Future;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;

import com.arquitecturajava.ejb.ServicioEJB1;
import com.arquitecturajava.ejb.ServicioEJB2;

@WebServlet("/ServletEJB")
public class ServletEJB extends HttpServlet {
        private static final long serialVersionUID = 1L;
        @EJB
        ServicioEJB1 servicio1;
        @EJB
        ServicioEJB2 servicio2;
        protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
                Future<String> futuro1=servicio1.getMensaje1();
                Future<String> futuro2= servicio2.getMensaje2();
                while(!futuro1.isDone() && !futuro2.isDone()) {
                        try {
                                Thread.sleep(6000);
                        } catch (InterruptedException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
                PrintWriter pw=response.getWriter();
                try {
                        pw.println(futuro1.get());
                        pw.println(futuro2.get());
                } catch (InterruptedException e) {
                        e.printStackTrace();
                } catch (ExecutionException e) {
```

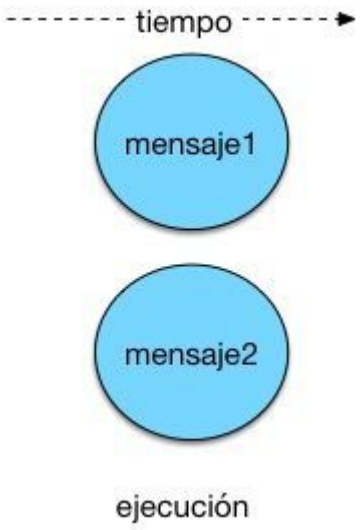
```
                    e.printStackTrace();
                }
        }

}
```

En este caso recibimos cada uno de los Futures creados por los EJBs. Usamos un bucle while y comprobamos el método isDone() de cada future para saber si ambas tareas han finalizado. En caso de no haber terminado ponemos el Servlet a dormir.

```
while(!futuro1.isDone() && !futuro2.isDone()) {
 .. sleep
}
```

Cuando ambos EJBs asíncronos han terminado su ejecución los métodos isDone() devolverán true. En esa situación ya podemos acceder a los valores futuros e imprimirlos por la pantalla.

De esta forma conseguiremos que aunque ambos EJBs tardan 5 segundos en ejecutarse las peticiones se ejecuten en paralelo y en 6 segundos hayamos terminado.



Otros artículos relacionados: EJB Async , EJB Remote vs Local (JEE 6), Introducción a EJB 3.1 (I)