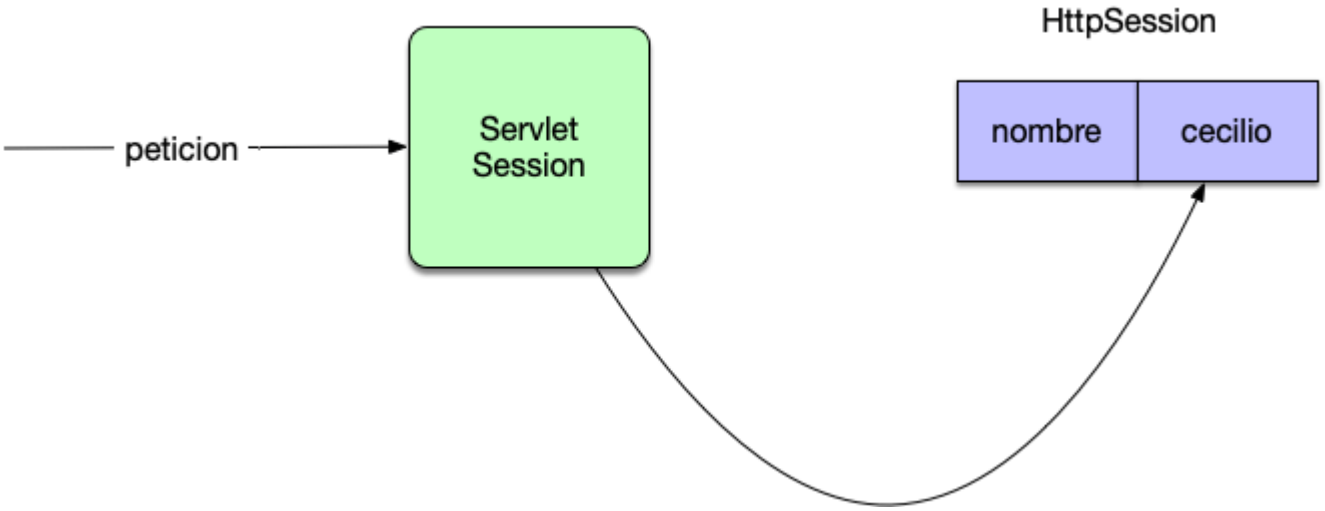
El uso de HttpSessionListener es muy práctico ya que nos permite ligar funcionalidad al propio ciclo de vida del manejo de sesiones . Vamos a ver unos ejemplos de como utilizarlo. Para ello el primer paso va a ser construir un Servlet que genere una sesión y nos permita almacenar algunos datos en ella.



Veamos su código :

```
import java.io.IOException;
import java.io.PrintWriter;

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 javax.servlet.http.HttpSession;

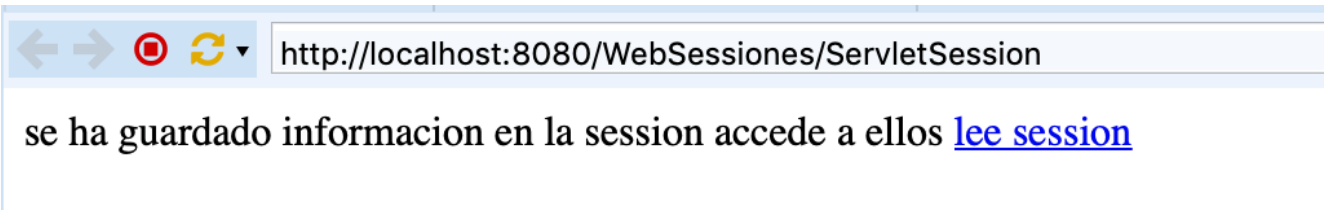
/**
```

```
 * Servlet implementation class ServletSession
 */
@WebServlet("/ServletSession")
public class ServletSession extends HttpServlet {
	private static final long serialVersionUID = 1L;

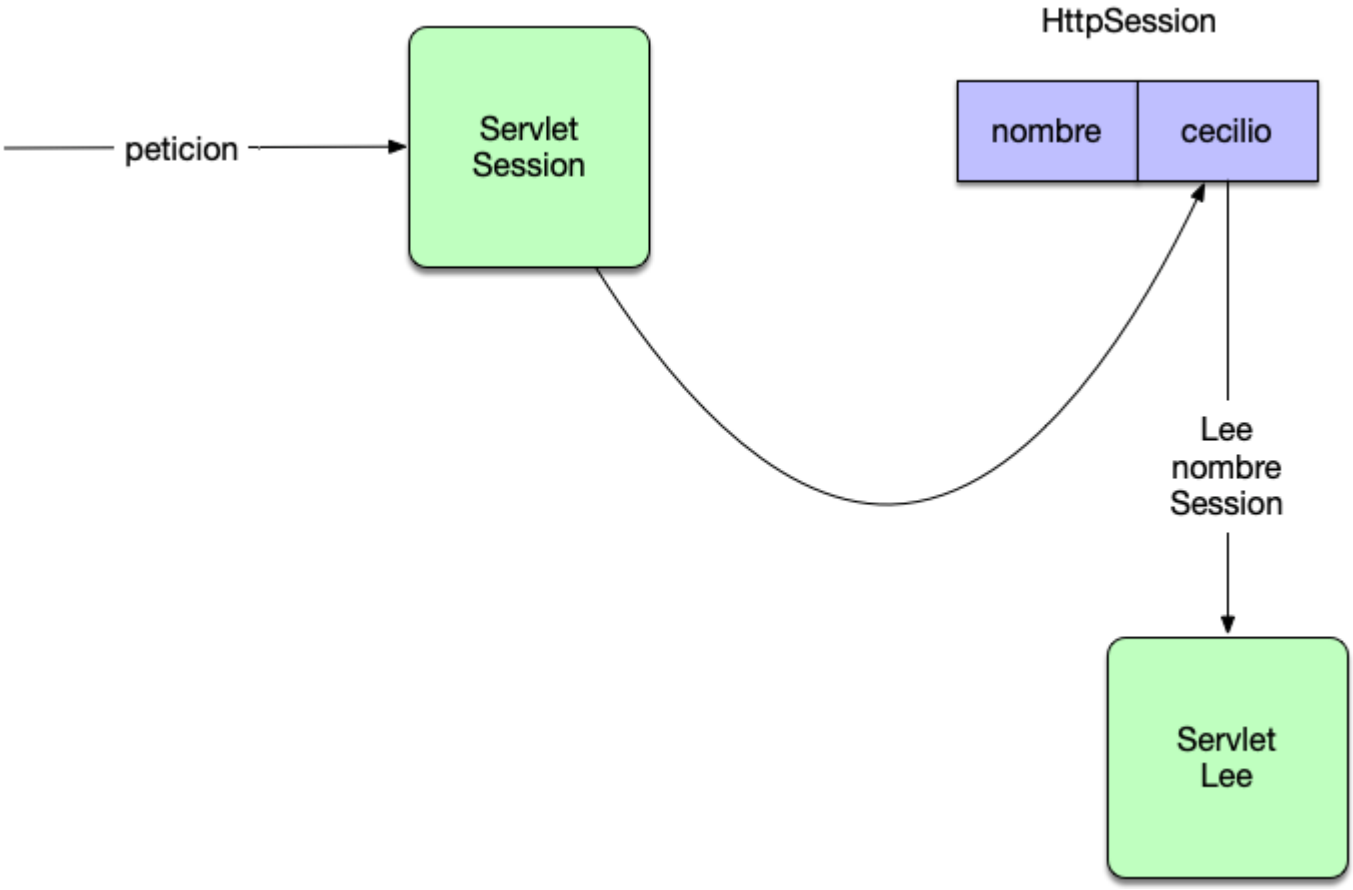
	protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
	HttpSession misession= request.getSession();
	misession.setAttribute("nombre", "cecilio");
	PrintWriter pw= response.getWriter();
	pw.println("<html><body>se ha guardado informacion en la
session accede a ellos");
	pw.println("<a href='ServletLeeSession'>lee
session</a></body></html>");
	}


}
```

Una vez tenemos construido el servlet que nos almacena la información en sesion le invocamos y nos presentará una página sencilla a nivel del navegador:



El siguiente paso es crear un Servlet que nos lea esa información y acceder a ella.



Veamos su código:

```
package com.arquitecturajava;
package com.arquitecturajava;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
```

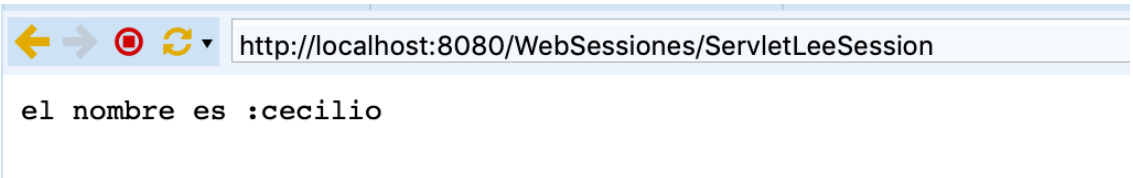
```
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;


@WebServlet("/ServletLeeSession")
public class ServletLeeSession extends HttpServlet {
        private static final long serialVersionUID = 1L;
        protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
                HttpSession misession= request.getSession();
                String nombre=(String)
misession.getAttribute("nombre");
                PrintWriter pw= response.getWriter();
                pw.println("el nombre es :"+nombre);
        }



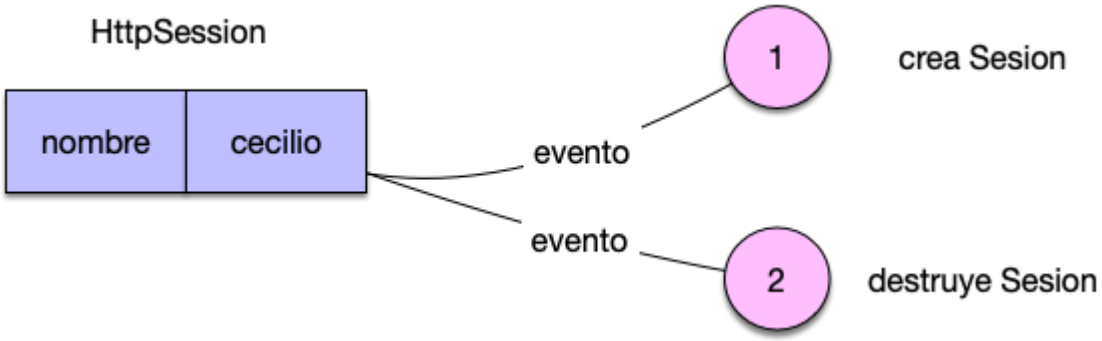
}
```

En este caso en cuanto pulsemos al link del primer Servlet se ejecutará el segundo y accederemos a los datos ubicados en la sesión.



Utilizando HttpSessionListener

Este es la forma más habitual de trabajar , sin embargo siempre quedan situaciones en las que necesitamos un control más a detalle de las sesiones a nivel de su propio ciclo de vida .



Por ejemplo suele ser muy interesante saber cuando iniciamos una sesión y cuando esta sesión se destruye. Para ello deberemos usar un HttpSessionListener. Vamos a ver como podemos usarlo , el primera paso es generar un web.xml y reducir el tiempo de timeout de sesión para que nos sea cómodo procesarlo.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
  <display-name>WebSessiones</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
```

```
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
  <session-config>
  <session-timeout>1</session-timeout>
  </session-config>
</web-app>
```

Una vez hecho esto simplemente nos queda construir un listener que se encargue de estar atento a los eventos de inicio y fin de sesión a nivel de nuestra aplicación:

```
package com.arquitecturajava;

import javax.servlet.annotation.WebListener;
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;
@WebListener
public class MiListener implements HttpSessionListener {


    public void sessionCreated(HttpSessionEvent se)  {
        System.out.println("se ha creado una session leyendo info de
la base de datos");
    }

    public void sessionDestroyed(HttpSessionEvent se)  {
        String nombre=(String)se.getSession().getAttribute("nombre");
        System.out.println("guardando en la base de datos" +nombre);
```

```
    }
}
```

Una vez construida la clase no nos queda nada más que volver a iniciar el Tomcat y probar que la aplicación funciona correctamente , veremos cómo los eventos de sesión se imprimen en la consola cuando la sesión se crea y se destruye. Estos podrían ayudarnos a cargar datos de la base de datos o ha almacenarlos cuando la sesión se inicie o cuando esta expire.

```
dic. 28, 2018 9:36:38 A. M. org.apache.catalina.startup.Catalina star
INFORMACIÓN: Server startup in [517] milliseconds
se ha creado una session leyendo info de la base de datos
guardando en la base de datoscecilio
```


1. HttpSession invalidate y sus problemas
2. Usando Java Session en aplicaciones web
3. Java ServletContextListener
4. Oracle HttpSession