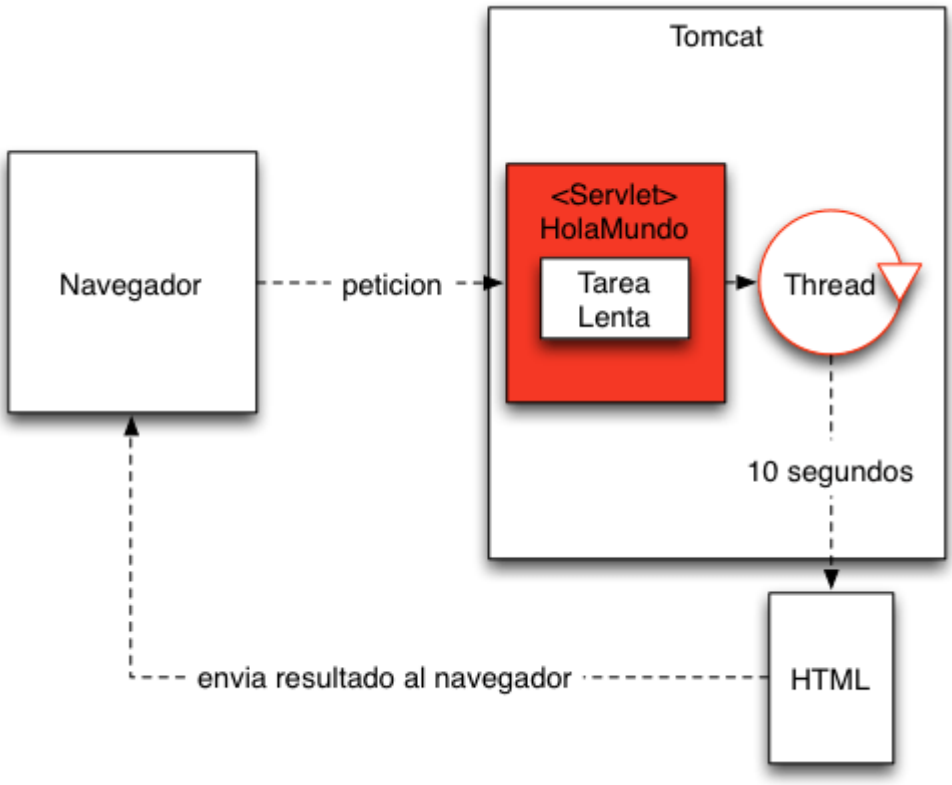
En el articulo anterior hemos usado las anotaciones de Servlets 3.0 para dar de alta un servlet sin tener la necesidad de hacer uso del web.xml.En este articulo introduciremos el concepto de servlet asincrono . Supongamos que disponemos del siguiente servlet.

```
package com.arquitecturajava;

//omitimos imports

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

 protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

 PrintWriter pw= response.getWriter();
 pw.println("<html>");
 pw.println("<body>");

 for (int i = 0; i < 10; i++) {
 try {
 Thread.sleep(1000);
 } catch (InterruptedException e) {

 e.printStackTrace();
 }
 }
```

```
 pw.println("hola mundo");
 pw.println("</body>");
 pw.println("</html>");
 }
}
```

Como se puede ver el servlet tiene un buble for que le obliga a dormir durante un total de 10 s simulando una tarea que consume recursos y lleva bastante tiempo ejecutar.



Al tardar 10 segundos en ejecutar la tarea el usuario no recibirá información alguna sobre que es lo que ha sucedido y le dará la sensación que la aplicación ha fallado.Para evitar esta situación podemos construir un servlet asincrono .Este nuevo tipo de servlet se declara de la siguiente manera.

```
package com.arquitecturajava;

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

import javax.servlet.AsyncContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "HolaMundo", urlPatterns = { "/HolaMundo" },
asyncSupported = true)
public class HolaMundo extends HttpServlet {

private static final long serialVersionUID = 1L;

protected void doGet(HttpServletRequest request,
 HttpServletResponse response) throws ServletException, IOException {
 final PrintWriter pw = response.getWriter();
 pw.println("<html>");
 pw.println("<body>");
 pw.println("hola mundo asincrono tarea realizandose en background");
 System.out.println("Hilo Principal:" +
Thread.currentThread().getName());
 final AsyncContext contextoAsincrono = request.startAsync();
 contextoAsincrono.setTimeout(12000);

contextoAsincrono.start(new Runnable() {
 @Override
```

```
 public void run() {
 for (int i = 0; i <= 10; i++) {
 System.out.println("Hilo Tarea Asincrona :"
 + Thread.currentThread().getName());
 try {
 Thread.sleep(1000);
 } catch (InterruptedException e) {
 }
 }
 contextoAsincrono.complete();

}
 });
 pw.println("</body>");
 pw.println("</html>");
 pw.close();

}

}
```

Es muy similar a un servlet clásico con la peculiaridad de que soporta un nuevo atributo “asyncSupported = true”. Vamos a comentar mas a detalle el siguiente bloque de código del método doGet.

```
pw.println("hola mundo asincrono tarea realizandose en background");
System.out.println("Hilo Principal:" +
Thread.currentThread().getName());
final AsyncContext contextoAsincrono = request.startAsync();
contextoAsincrono.setTimeout(12000);
```

```
contextoAsincrono.start(new Runnable() {
 @Override
 public void run() {
 for (int i = 0; i <= 10; i++) {
 System.out.println("Hilo Tarea Asincrona :"
 + Thread.currentThread().getName());
 try {
 Thread.sleep(1000);
 } catch (InterruptedException e) {
 }
 }
 contextoAsincrono.complete();

}
 });
```

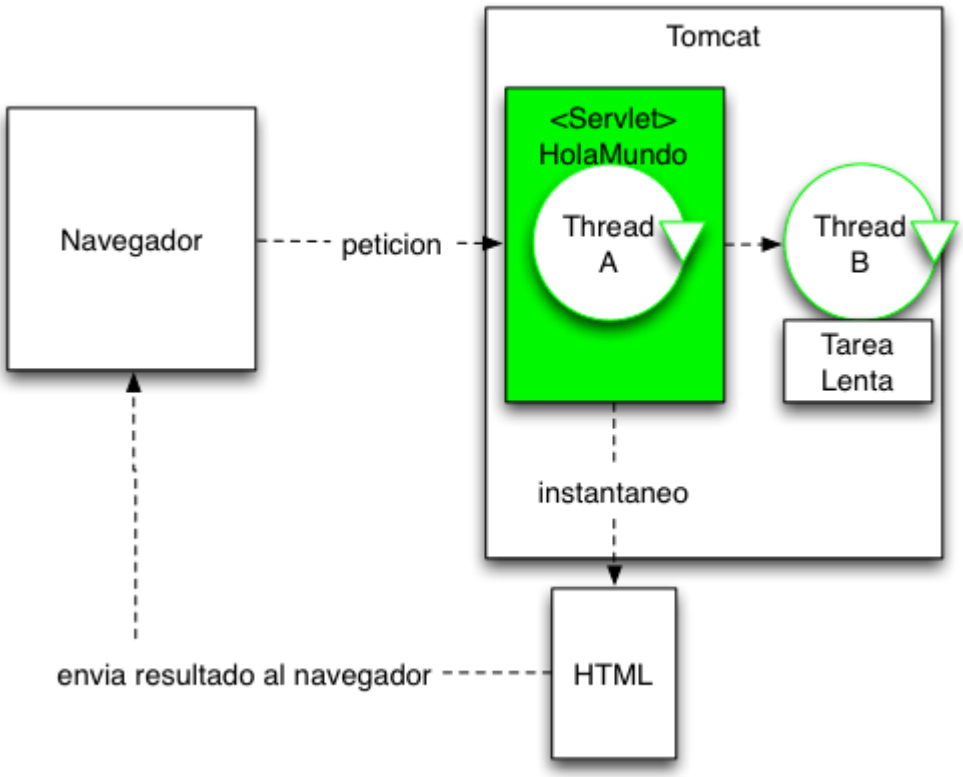
El método en cuestión modifica la forma de ejecutar el servlet a traves de la siguiente linea de código

```
final AsyncContext contextoAsincrono = request.startAsync();
```

Esta linea define un contexto de ejecución asincrono para que mas adelante podemos pasar a este contexto un objeto de tipo Runnable (para lanzar un thread en paralelo)

```
contextoAsincrono.start(new Runnable() {
```

Una vez hechas estas dos operaciones ,el contexto lanzara un nuevo thread para ejecutar la tarea que nosotros le hemos solicitado de forma asincrona. Sin penalizar al thread actual que esta ejecutando el método doGet del servlet



A continuación podemos ver en la consolo de Tomcat como el Servlet se ejecuta en el hilo principal (hilo 5)y la tarea asincrona en otro hilo (hilo 6).

```
INFO: Server startup in 5706 ms
Hilo Principal:http-bio-8080-exec-5
Hilo Tarea Asincrona :http-bio-8080-exec-6
Hilo Tarea Asincrona :http-bio-8080-exec-6
Hilo Tarea Asincrona :http-bio-8080-exec-6
Hilo Tarea Asincrona :http-bio-8080-exec-6
Hilo Tarea Asincrona :http-bio-8080-exec-6
Hilo Tarea Asincrona :http-bio-8080-exec-6
Hilo Tarea Asincrona :http-bio-8080-exec-6
Hilo Tarea Asincrona :http-bio-8080-exec-6
Hilo Tarea Asincrona :http-bio-8080-exec-6
Hilo Tarea Asincrona :http-bio-8080-exec-6
Hilo Tarea Asincrona :http-bio-8080-exec-6
```