http://www.egovframe.go.kr/wiki/doku.php?id=egovframework:rte2:ptl:async_request

Asynchronous request processing

Outline

In the previous version, the number of threads always matches the number of requests. Threads stood by until the request is out and responded, giving rise to limited processing of request for response.

Asynchronous Request Processing of Servlet 3.0 causes the Servlet Thread returned after request, followed by another Thread coming into play for processing. The Servlet Thread will then be back to keep on request processing.

The minimum requirements for asynchronous request processing are as follows:

- Servlet 3.0 or newer;
- Spring MVC 3.2 or newer (included in eGov 3.0)
- Configurations in Web.xml

How to change Pom Dependency for Servlet 3.0

```
<!-- Servlet -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.0.1</version>
    <scope>provided</scope>
</dependency>
```

Configurations in Web.xml

It is advised that the Servlet version in web.xml shall be configured as follows:

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
      version="3.0">

      ...

</web-app>
```

You'll then need to configure Tag Async-supported of Servlet Configuration in web.xml for True:

```
<servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
    <async-supported>true</async-supported>
</servlet>
```

Description

In asymmetrical request processing for Spring, the Servlet Thread causes a request processed before the Class Async Provision for Spring comes into play to activate asymmetrical mode to return the Servlet Thread back, followed by service processing by the thread of internal application. Followed by that is the Servlet Thread receiving the response back to transmit to the client.

The Class provided by the asymmetrical request processing for Spring is classified by the type of thread, based upon which internal service is processed after return of the Servlet Thread.

Callable

When the Servlet Thread for request processing is returned back, it gets asymmetrical by the thread controlled by Spring MVC.

Processing by the Callable is as follows:

1. Callable Object is returned by the Method RequestMapping of Controller instead of the Object View or String.
2. The Servlet Thread is returned back along with Callable, thereby delegating asymmetrical processing to Callable.
3. Asyncronization is done in the Thread governed by TaskExecutor in Spring MVC.
4. The value returned back from the Function call() in Callable is transferred back to the Servlet Thread.

Callable is preferred when you work the DB, REST API request or any other tasks that accompanies extensive request processing.

```
@RequestMapping(“/view”)
public Callable<String> callableWithView(final Model model) {
   return new Callable<String>() {
      @Override
      public String call() throws Exception {
         Thread.sleep(2000);
         model.addAttribute(“foo”, “bar”);
         model.addAttribute(“fruit”, “apple”);
         return “view”;
      }
   }
```

DefferedResult

Asymmetrical request is processed by the Thread not governed by Spring MVC after returning the Servlet Thread back. DefferedResult is most commonly used in JMS, AMQP, Scheduler, Redis, and other HTTP requests.

Processing by the DefferedResult is as follows:

1. Controller returns DefferedResult and saves DefferedResult in In-Memory Queue or List.
2. When the Servlet Thread is returned, the Object DefferedResult is made available in Queue when an evet occurs.

```
@RequestMapping("/quotes")
@ResponseBody
public DeferredResult<String> quotes() {
   DeferredResult<String> deferredResult = new DeferredResult<String>();
   // Save the deferredResult in in-memory queue ...
   queue.add(defferedResult);

   return deferredResult;
}

// In some other thread...
@RequestMapping("/someEvent")
@ResponseBody
public String someEvent(String data) {
```

```
    for(DefferedResult<String> result : queue) {
        result.setResult(data);
    }

    return "view";
}
```

AsyncTask

AsyncTask works basically the same with Callable, whereby Controller contains Callable when returned.

You may add Timeout, designate AsyncTaskExecutor or take Thread Pool apart from the rest, if desired.

```
@RequestMapping("/facebooklink")
public WebAsyncTask<String> facebooklink() {
    return new WebAsyncTask<String>(
        30000L, // Timeout
        "facebookTaskExecutor", // TaskExecutor
        new Callable<String>() {
            @Override
            public String call() throws Exception {
            // Job
                return result;
            }
        }
    );
}
```

References

- Spring Framework - Reference Document / 4.1 Support for Servlet 3 based asynchronous request processing
- Spring Framework - Reference Document / 17.3.4 Asynchronous Request Processing