**Asynchronous JAX-RS**

1. Purpose of Asynchronous JAX-RS

2. Server Asynchronous Response Processing (or) Developing Asynchronous JAX-RS Resource

2.1 Use Case Related to need of Asynchronous JAX-RS API

2.2 Configuring Vendor Specific Runtime to work with Async REST Resource

|-Configuring Async RESTEasy Runtime

|-Configuring Async JERSEY Runtime

2.3 Developing Asynchronous Resources using JAX-RS API (or) AsyncResponse API

|-Developing the Async JAX-RS Resource

|-Exception Handling

|-Cancel

|-Status Methods

|-Timeouts

|-Callbacks

|-Use Cases for AsyncResponse

3. Asynchronous JAX-RS Client (or) AsyncInvoker Client API

3.1 Purpose of Async JAX-RS Client

3.2 Different ways of Asynchronous client Invocations

|-Using Futures

|-A different rare architectures (Specific to some use cases only)

|-Exception handling

|-Using Callbacks

3.3 Futures Versus Callbacks

|-Wrapping Up

**1. Purpose of Asynchronous JAX-RS:**

Whenever we have moderate or more complicated business logic which is going to take more amount of time then we should not use Synchronous JAX-RS API rather we need to use Asynchronous JAX-RS Resources that has been provided by the JAX-RS API so that scalability of the application will improves and in addition to this we will get de-coupling between the components.

The JAX-RS has a support for to develop the both Synchronous Resources and Aynsnchronous Ressources bcz JAX-RS API support for Asynchronous Programming. But with JAX-RS API we will not get Asynchronous Communication and it will never supports for the Asynchronous Communication.

Another interesting new feature introduced in JAX-RS 2.0 is asynchronous request and response processing both on the client and server side. If you are mashing together a lot of data from different websites or you have something like a stock quote application that needs to push events to hundreds or thousands of idle blocking clients, then the JAX-RS 2.0 asynchronous APIs are worth looking into.

**2. Server Asynchronous Response Processing (or) Developing Asynchronous JAX-RS Resource:**

**2.1 Use Case Related to purpose of Asynchronous JAX-RS Resources:**

For a typical HTTP server, when a request comes in, one thread is dedicated to the processing of that request and its HTTP response to the client. This is fine for the vast majority of HTTP traffic both on the Internet and on your company's internal networks.

Most HTTP requests are short-lived, so a few hundred threads can easily handle a few thousand concurrent users and have relatively decent response times.

The nature of HTTP traffic started to change some-what as JavaScript clients started to become more prevalent. One problem that popped up often was the need for the server to push events to the client.

Stock Quote Application:

A typical example is a stock quote application where you need to update a string of clients with the latest stock price. These clients would make an HTTP GET or POST request and just block indefinitely until the server was ready to send back a response. This resulted in a large amount of open, long-running requests that were doing nothing other than idling. Not only were there a lot of open, idle sockets, but there were also a lot of dedicated threads doing nothing at all. Most HTTP servers were designed for short-lived requests with the assumption that one thread could process requests from multiple concurrent users. When you have a very large number of threads, you start to consume a lot of operating system resources. Each thread consumes memory, and context switching between threads starts to get quite expensive when the OS has to deal with a large number of threads. It became really hard to scale these types of server-push applications since the Servlet API, and by association JAX-RS, was a "one thread per connection" model.

In 2009, the Servlet 3.0 specification introduced asynchronous HTTP or asynchronous Programming. With the Servlet 3.0 API, we can suspend the current server-side request and have a separate thread,

other than the calling thread, handle sending back a response to the client. That menas now Server will have 2-thread pools

1. Request Thread Pool

2. Response Thread Pool

Till Servlet 2.0 the server is called as "one thread per connection" bcz it has only one Thread Pool, But in from Servlet 3.0 onwards server has provided 2-thread pools to avoid the problems that are there in the "one thread per connection" architectured server.

For a server-push application like Stock management, we could then have a small handful of threads manage sending responses back to polling clients as like Servlet 3.0 the JAX-RS has provided API to avoid all the overhead or problems of the "one thread per connection" model. That is the reason JAX-RS 2.0 introduced Async JAX-RS API to develop the Asynchronous JAX-RS Respurces so that it can be deployed in Asynchrnous supported processing servers.

In order to implement this Asynchronous RESTful the Implementaion vendors also need to provide a diferent vendor specific Runtime (may provide different Runtimes for Sync and Async or they may use same Runtime to work with both) to work with Asynchronus Resource development and along with this runtime we need to turn ON the Asynchronous programing in the server stating that our Resource is Asynchronus Resource that where we need to turn ON the Asynchronous programming in the server as well by configuring our Runtime Servlet in the web.xml as

<servlet>

<!-- Configure Vendor specific Runtime Servlet-->

<async-supported>true</async-supported>

</servlet>

The <async-supported>true</async-supported> tag is newly added in Servlet 3.0

So now our Runtime is Async and Server also turn ON's the Async mode of req processing.

Can u plz tell any new tags are added in web.xml?

Note:

Server-side async response processing is only meant for a specific small subset of applications. Asynchronous doesn’t necessarily mean automatic scalability. For the typical web app, using server asynchronous response processing will only complicate our code and make it harder to maintain. It may even hurt performance.

**2.2 Configuring Vendor Specific Runtime to work with Async REST Resource:**

|-Configuring Async RESTEasy Runtime

|-Configuring Async JERSEY Runtime

Configuring Async RESTEasy Runtime:

Whenever the client sends the req the JAX-RS Runtime which is specific to the vendor that menas RESTEasy Runtime will takes the req but in order to work with Asnyc programming Resources the "HttpServletDispatcher" will not works bcz it is Synchronus Servlet hence the RESTEasy has provide one more Servlet called as "HttpServlet30Dispatcher" which is Asynchronous Servlet bcz it has been annotated with @WebServlet(@WebServlet(asyncSupported = true)) so that Servlet Container will understoods it is an Async Servlet which complient with Servlet 3.0 specification that is the reason they named as "30".

So we need to configure "HttpServlet30Dispatcher" as Servlet and we need to turn ON the Async programming to the server in web.xml with in the servlet.

class HttpServlet30Dispatcher extends HttpServletDispatcher {

}

class HttpServletDispatcher extends HttpServlet {

}

web.xml

<web-app>

.....

<servlet>

<servlet-name>AsyncResteasy</servlet-name>

<servlet-class>org.jboss.resteasy.plugins.server.servlet.HttpServlet30Dispatcher

</servlet-class>

.....

<async-supported>true</async-supported>

</servlet>

</web-app>

Configuring Async JERSEY Runtime:

The JERSEY Implementaion has provided same Runtime "org.glassfish.jersey.servlet.ServletContainer" to work with both Sync and Asyn Resource developement but the difference is that if we turn ON the Async programming so that it will works as Async if we turn OFF the Async mode it will works as Sync mode.

web.xml

<web-app>

....

<servlet>

<servlet-name>AsyncJersey</servlet-name>

<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>

.....

<async-supported>true</async-supported>

</servlet>

</web-app>