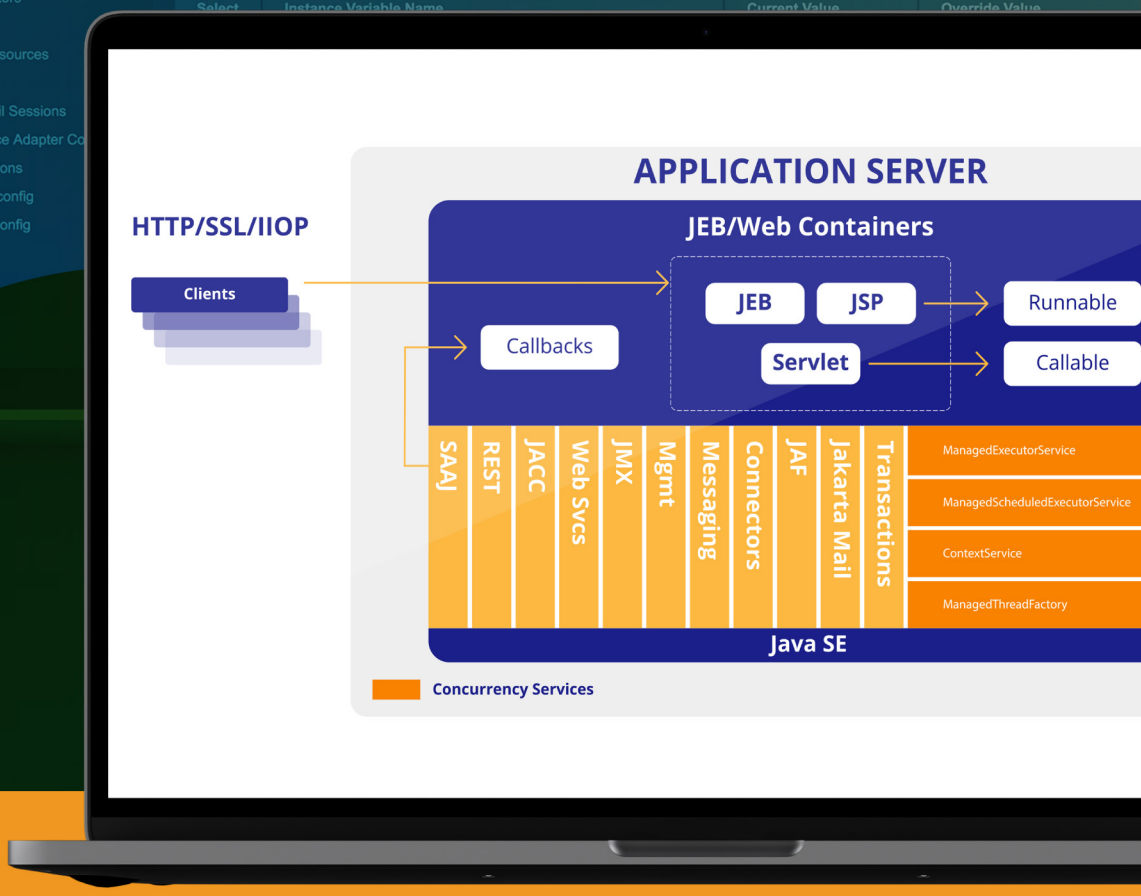
Jakarta Concurrency: All You Need to Know



The Payara® Platform - Production-Ready, Cloud Native and Aggressively Compatible.

Fact Sheet

Contents

Introduction

Jakarta EE, previously Java EE, is a set of specifications that enables the worldwide community of Java developers to work on cloud native Java enterprise applications. It is an open source project maintained by the Eclipse Foundation.

Jakarta Concurrency is one of its key specifications, which concentrates on user-defined asynchronous execution.

Why Do We Need Jakarta Concurrency?

Programming in Jakarta EE mostly frees a developer from thinking about threads, parallelism, synchronization, etc. It happens automatically in most use cases with satisfactory results and performance. All threads are owned by the application server, and it can effectively manage them, pre-allocate, or remove on shutdown.

The situation changes when the amount of work to deliver is large – for example, creating huge reports or processing many REST calls. There are a limited number of threads for processing events, and they can be exhausted. Asynchronous processing is here to help.

Jakarta Concurrency is an alternative to the `java.util.concurrency` package known to any experienced Java programmer.

Code processing data in a separate thread needs to know the context in which it is executed. Which user is logged in? What are his rights? What is the data stored in the session? What is the classpath of the calling module? What are the running transactions? All that information is stored in the provided context.

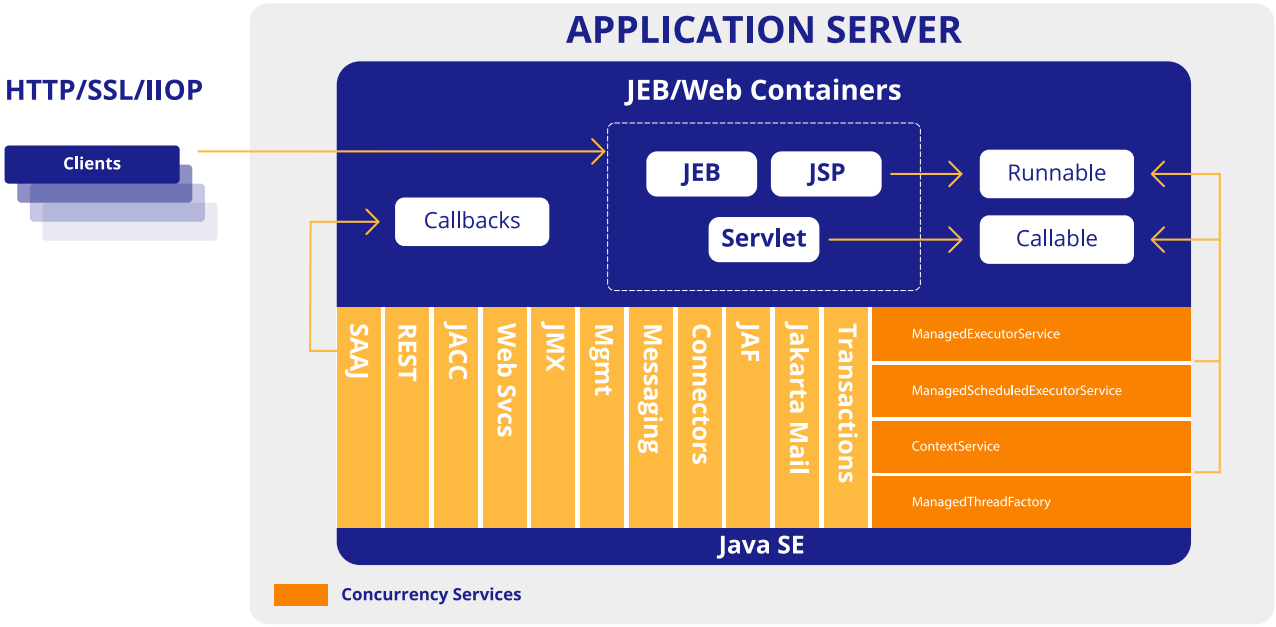
A bit more advanced question is: “What data source is set in the calling module?” Yes – the setup can be set per component (JNDI `"java:comp/"` name) and still use shared code; the evaluation respects the calling context.

What is the Advantage Compared to Java SE Threads?

Jakarta Concurrency is highly configurable. The big advantage is that the administrator can accommodate the behavior of a final application to the hardware running the server. The thread pool can be set to one thread when running on a small server or to hundreds of threads for “Big Iron” (an extremely large, expensive and fast computer) with hundreds of CPU cores.

What are the Main Components?

Concurrency is accessible by all other specs, as explained in the image from the Concurrency specification. See the orange-colored part of the diagram!



Source: https://jakarta.ee/specifications/concurrency/3.0/jakarta-concurrency-spec-3.0.html#-container-thread-context

Managed Executor Service

`ManagedExecutorService` (or MES) is the key component, allowing parallel execution with the context. This example runs `BackService.longJob()` method in a new thread, sharing context, including security:

```
@Path("jobs")
public class JobsResource {
    @Resource
    private ManagedExecutorService managedExecutor;
    @EJB
    private BackService backService;
    @GET
    @Path("longJob")
    public String longJob() {
        managedExecutor.submit(() -> {
            backService.longJob();
        });
        return "Job Submitted";
    }
}
```

A similar service is `ManagedScheduledExecutorService`, allowing scheduling repeating tasks.

Managed Thread Factory

`ManagedThreadFactory` is useful for third party libraries, which are creating threads but are not part of Jakarta EE. They usually allow you to pass `ThreadFactory` as a parameter.

It can also be used for more explicit work with threads.

Context Service

When a third party library creates threads, but does not accept `ThreadFactory` as a parameter, it is necessary to wrap the `Runnable` using the `ContextService`. It creates a proxy, which sets the correct context before the code is executed.

Resource Configuration

All resources (`ManagedExecutorService`, `ManagedThreadFactory`, `ContextService`) can be configured in xml accompanying the application or by the application server.

```
<managed-executor>
    <name>java:app/concurrent/myExecutor</name>
    <max-async>3</max-async>
</managed-executor>
```

What is New in Jakarta EE 10 / Concurrency 3.0?

@Asynchronous

One of the most useful new features is the annotation `@Asynchronous`, which can be now used on a method of a POJO class or on a class itself.

With `@Asynchronous`, the previous example can be simplified:

```
@Path("jobs")
public class JobsResource {
    @EJB
    private BackService backService;

    @GET
    @Path("longJob")
    public String longJob() {
        backService.longJob();
        return "Job Submitted";
    }
 }
```

In the definition of the `BackService`, the asynchronous method must return `Asynchronous.Result`:

```
public class BackService {
   @Asynchronous
   public CompletableFuture<Result> longJob() {
      Result result = processLongJob();
      return Asynchronous.Result.complete(result);
   }
}
```

The asynchronous methods must return type `CompletableFuture<>` and end with `return Asynchronous.Result.complete(result);`

Definition of Resources by Annotation

So far, all resources like thread pools or managed executor services (other than the default ones) need to be defined on the server prior to deployment. This changes with the new annotation, allowing us to define these resources in the code. They are then created dynamically during deployment.

Let us configure the behavior of `@Asynchronous` by `ManagedExecutorService` with up to three threads and name `myExecutor`. All methods in this case share the same asynchronous behavior:

```
@ManagedExecutorDefinition(name = "java:app/concurrent/myExecutor", maxAsync =
3)
@Asynchronous (executor = "java:app/concurrent/myExecutor")
public class BackService {
```

The same configuration is available for `ManagedThreadFactory` and `ContextService`.

Adding Context to java.util.concurrent

The ForkJoin framework became popular with the usage of Java streams and `CompletableFuture`. Concurrency 3.0 adds support for it, `ManagedThreadFactory` extends `ThreadFactory`, ForkJoinWorkerThreadFactory and can be used for streams. The example executes the map in parallel, all thread with the Jakarta EE context:

```
@Resource
ManagedThreadFactory threadFactory;
ForkJoinPool fj = new ForkJoinPool(4, threadFactory, null, false);
fj.submit(() -> {
    return Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9)
   .parallelStream()
  .map(...)
```

Support for `CompletableFuture` is similarly seamless. All the methods get the context and are executed synchronously or asynchronously as required.

```
backService.longJob()
     .thenApply(method1())
     .thenApplyAsync(method2())
     .thenApply(method3())
     .thenApplyAsync(method4())
    .get();
```

Thread Context Providers

Thread context providers solve the problem of sharing data between threads. It offers a well-defined way to pass data from the calling thread to the new one and back. More information can be found in the Concurrency documentation.

Top Tips

It is easy to start with Concurrency in Jakarta EE. If you have a code, which can benefit from parallel processing, start with injecting `ManagedThreadFactory`, and start `parallelStream()`!

If you are tired of configuring your own `ManagedExecutorService` or you don't want to use it at all, it is time to configure it by the annotation and enjoy the power of multiple threads.

And if you missed using asynchronous CompletableFuture, because it was not executed in the same JPA transaction, now it is supported.

Payara has a set of examples in Payara samples/concurrency.

Many advanced examples can be found in the Concurrency TCK, they work in any compatible implementation including Payara Platform 6 Community.

Get Started with Jakarta Concurrency 3.0 and Payara Platform

Payara Platform is an innovative and supported application server, ideal to facilitate and enhance your Jakarta EE and MicroProfile projects.

Payara Platform 6 Community is the latest major release of the Platform, and is officially compatible with Jakarta EE 10. This means you can try out Jakarta Concurrency 3.0 immediately by downloading Payara Platform 6 Community here. This is our product for learning and innovation, whilst Payara Server Enterprise is designed for mission-critical projects. Payara Platform 6 Enterprise will have

tooling to support migrating applications and Payara domains from Jakarta EE 8 to Jakarta EE 10, and is coming very soon. Request Payara Server Enterprise here.

Further Reading

Jakarta EE CDI Fact Sheet

Jakarta EE 10: What Decision Makers Need to Know

Dismiss the Myths: Get to know Jakarta EE (Java EE) eBook

sales@payara.fish

UK: +44 800 538 5490
Intl: +1 888 239 8941

www.payara.fish