Spring Boot 3.0 and Spring Framework 6.0 Features

Description

Introduction

In this blog post we are going to see some of the most interesting and new features introduced in spring Boot 3 along with Spring Framework 6.0. This Spring Boot 3 was released in November 2022, which is one of the major releases after Spring boot 2.0. Being a developer we need to aware of all the features which will make our life easy. One of the surprising and shocking decision made by the makers of Spring, was to drop the support for older versions of Java. So by the end of this post I guarantee that you will get enough knowledge and you will make use of full potential that Spring Boot 3 and Spring framework 6.0 provides. Lets get started.

Features of Spring Boot 3.0 and Spring framework 6.0

Minimum Java Version required for Spring Boot 3

Java 17 which is the LTS version becomes the minimum baseline version that will be required by Spring Boot 3. This does not mean that Java 17 was not supported before, it was supported before also. Any Spring Boot 2.x release will work perfectly fine with Java 17. You can also leverage the benefits of Java 17 features (such as records) in your own codebases.

Records in Spring Boot

The purpose of introducing Java records (JEP 395) was because it is quick way to create data carrier classes, i.e. the classes whose primary moto is to simply contain data and carry it between modules, also known as POJOs (Plain Old Java Objects) and DTOs (Data Transfer Objects).

```
public record Employee (String name, Integer age) {}
```

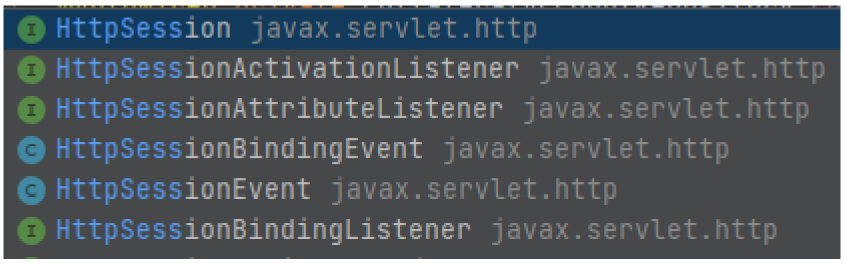
Before implementing the records, be careful if it satisfies your bean validation requirements or whether you need to write custom validation.

New Jakarta package in Spring Boot

Jakarta EE 9 which is a new top-level **jakarta** package has replaced EE 8â€™s **javax** top-level package.

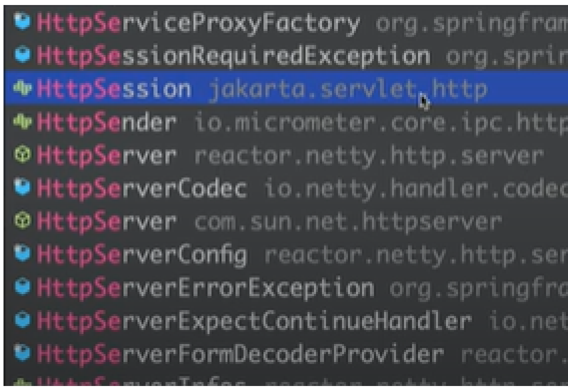
Most of the classes have been moved from javax package to jakarta package. So this is the biggest change we need to do when we move from Spring Boot 2 to Spring Boot 3. As you can see in the below image prior to Java 17 we had HttpSession inside javax.* package, but post Spring Boot 3, it has been moved to jakarta.* package.

Before Java 17



```
HttpSession javax.servlet.http
HttpSessionActivationListener javax.servlet.http
HttpSessionAttributeListener javax.servlet.http
HttpSessionBindingEvent javax.servlet.http
HttpSessionEvent javax.servlet.http
HttpSessionBindingListener javax.servlet.http
```

In Java 17



```
HttpServiceProxyFactory org.springfram
HttpSessionRequiredException org.sprin
HttpSession jakarta.servlet.http
HttpSender io.micrometer.core.ipc.http
HttpServer reactor.netty.http.server
HttpServerCodec io.netty.handler.codec
HttpServer com.sun.net.httpserver
HttpServerConfig reactor.netty.http.ser
HttpServerErrorException org.springfra
HttpServerExpectContinueHandler io.net
HttpServerFormDecoderProvider reactor.
```

Another example is that Servlet specification in Jakarta EE 8 uses a **javax.servlet** package but this has been changed to **jakarta.servlet** in EE 9. Below are some of the additional migrated packages.

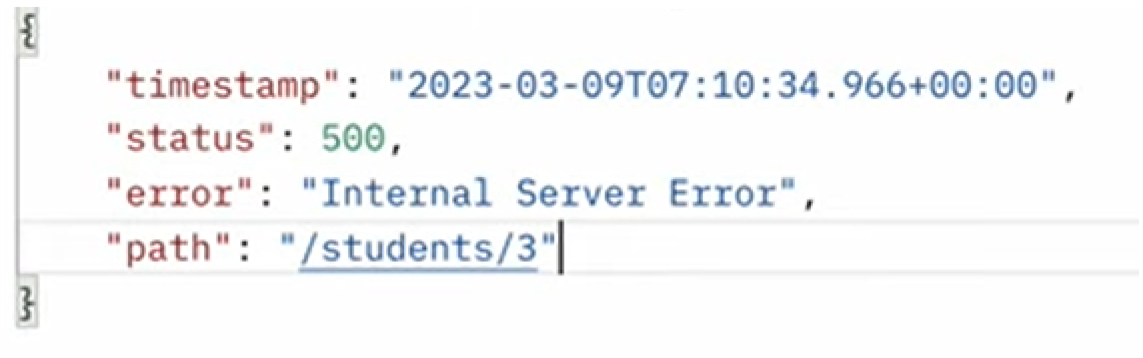
javax.persistence.*	**jakarta.persistence.***
javax.annotation.*	**jakarta.annotation.***
javax.validation.*	**jakarta.validation.***
javax.transaction.*	**jakarta.transaction.***

Note: Only the packages which were part of Java EE will change to Jakarta EE. Packages such as **javax.sql.*** and **javax.crypto.*** will not change because they were part of Java 17 JDK

Even if you are upgrading your application to latest Jakarta EE api's, we need to ensure that our code as well as third party libraries need to use **jakarta.*** package imports. Most of the third party libraries such as Thymeleaf, Hibernate, Tomcat, Jakarta Validation, Jetty, Hazelcast and Undertow have already done.

Problem Details

Whenever we have an exception thrown from our application most of us see below error.



```
{
    "timestamp": "2023-03-09T07:10:34.966+00:00",
    "status": 500,
    "error": "Internal Server Error",
    "path": "/students/3"
}
```

Every project and team within the industry has their own way of representing the exception object.

Usually we would create a ErrorMessage class, which would have the errorCode, errorMessage, status etc. In order to standardize the object, Spring boot has provided a ProblemDetails class.

Improved Observability with Micrometer and Micrometer Tracing

Observability is the ability to view the internal state of the running application. It consists of the three things i.e. logging, metrics and traces. For metrics and traces, Spring Boot uses Micrometer Observation. To create your own observations (which will lead to metrics and traces), you can inject an ObservationRegistry.

When our application is about to go live in production, then we focus more on how we can implement or improve the monitoring of our application. We take help of monitoring tools. If you want to capture the behavior of your application such as success responses, error responses in some monitoring tool. It is very easy to improve the observability in our application using Spring boot 3.0. Internally it is going to use micrometer, so with the help of actuator and Spring boot 3.0 we will be able to do it very easily.

Here is the official documentation from spring boot about Observability.

HttpExchange

Earlier for REST based communication, we were using REST clients, Web clients and we were also using Feign clients to invoke REST services or communicating between different services that we have in our microservices ecosystem. HttpExchange has been introduced by default which will allow us to define our REST API calls for different services using the declarative approach. In HttpExchange we will just define our interfaces and those interfaces will be responsible to define that we need to call this API.

Here is the official documentation from spring boot about HttpExchange.

Spring Boot 3.0 FAQ

What is Spring Boot 3 compatible with?

Spring Boot 3.0.5 requires Java 17 and is compatible up to and including Java 20

Which version of Spring Boot works with Java 11?

Spring Boot 2.4 supports Java 15 while also remaining compatible with Java 11 and 8.

Does Spring Boot 3 work with Java 11?

No. Spring Boot 3.0.0 requires JDK 17 as a minimum version to work with it. It is mentioned in the Spring Boot 3.0 release document. If you want to use Java version lower than Java 17, you will have to work with Spring Boot 2.0.

When was spring boot 3 released?

Spring boot 3 was released in November 2022.

Can I run Spring Boot 3 with Java 11?

Spring Boot 3.0 requires Java 17 as a minimum version. If you are currently using Java 8 or Java 11, you'll need to upgrade your JDK before you can develop Spring Boot 3.0 applications.

Should I upgrade to Spring Boot 3?

It is not recommended to upgrade directly from a version lower than Spring Boot 2.7 to Spring Boot 3.0, otherwise too many new features and API changes will require you to change a lot of configurations and the upgrade path will be too steep.

Summary

I hope you liked this post on Spring boot 3. Feel free to comment if you have any suggestions or improvements or you have any query related to any content on the website.

Category

1. Spring Boot

Date Created
March 26, 2023
Author
kk-ravi144gmail-com