Spring Boot 3.x introduces several significant changes and improvements compared to Spring Boot 2.x. Here are some of the key differences:

1. **Java Version Support**:
   * **Spring Boot 2.x**: Primarily supports Java 8 and Java 11.
   * [**Spring Boot 3.x**: Requires Java 17, which is a Long-Term Support (LTS) version1](https://www.baeldung.com/spring-boot-3-migration)[2](https://blog.csdn.net/lkp1603645756/article/details/139065816).
2. **Spring Framework**:
   * **Spring Boot 2.x**: Based on Spring Framework 5.x.
   * [**Spring Boot 3.x**: Built on Spring Framework 6, which includes support for Jakarta EE 91](https://www.baeldung.com/spring-boot-3-migration)[2](https://blog.csdn.net/lkp1603645756/article/details/139065816).
3. **Jakarta EE**:
   * **Spring Boot 2.x**: Uses Java EE APIs.
   * [**Spring Boot 3.x**: Migrates to Jakarta EE APIs, which involves package name changes from javax to jakarta1](https://www.baeldung.com/spring-boot-3-migration).
4. **Native Image Support**:
   * **Spring Boot 2.x**: Limited support for GraalVM native images.
   * [**Spring Boot 3.x**: Enhanced support for GraalVM native images, allowing applications to be compiled into native executables, which can significantly reduce startup time and memory usage](https://www.baeldung.com/spring-boot-3-migration)[2](https://blog.csdn.net/lkp1603645756/article/details/139065816).
5. **Configuration Changes**:
   * **Spring Boot 3.x**: Some configuration properties have been renamed or removed. [For example, spring.redis has moved to spring.data.redis, and server.max.http.header.size has moved to server.max-http-request-header-size1](https://www.baeldung.com/spring-boot-3-migration).
6. **Security Enhancements**:
   * [**Spring Boot 3.x**: Introduces stronger password storage mechanisms, more flexible security configuration options, and support for OAuth 2.1](https://www.baeldung.com/spring-boot-3-migration)[2](https://blog.csdn.net/lkp1603645756/article/details/139065816).
7. **Performance Improvements**:
   * [**Spring Boot 3.x**: Optimized for better startup time and reduced memory usage, especially beneficial for microservices and cloud-native applications](https://www.baeldung.com/spring-boot-3-migration)[2](https://blog.csdn.net/lkp1603645756/article/details/139065816).
8. **Dependency Updates**:
   * [**Spring Boot 3.x**: Updates to dependencies like Hibernate (to version 6.1.4) and JPA (to version 3.1), ensuring compatibility with the latest standards1](https://www.baeldung.com/spring-boot-3-migration).

These changes make Spring Boot 3.x a more modern, efficient, and secure framework compared to its predecessor. If you’re planning to migrate from Spring Boot 2.x to 3.x, it’s important to review these differences and update your application accordingly.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SessionManagement\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In Spring Boot 3.x, session management can be handled using various methods, including in-memory sessions, JDBC-based sessions, and Redis-based sessions. Here’s an overview of how to manage sessions in Spring Boot 3.x:

**1. In-Memory Sessions**

In-memory sessions store session data in the server’s memory. This is suitable for simple applications but not recommended for production due to scalability and persistence issues.

**2. JDBC-Based Sessions**

JDBC-based sessions store session data in a relational database. This approach is more scalable and persistent compared to in-memory sessions.

To configure JDBC-based sessions, add the following dependencies to your pom.xml:

**XML**

<dependency>

<groupId>org.springframework.session</groupId>

<artifactId>spring-session-jdbc</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

Then, configure the datasource in your application.properties:

spring.datasource.url=jdbc:mysql://localhost:3306/mydb

spring.datasource.username=root

spring.datasource.password=secret

Finally, enable JDBC session management in your configuration class:

**Java**

@Configuration

@EnableJdbcHttpSession

public class HttpSessionConfig {

// Additional configuration if needed

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**3. Redis-Based Sessions**

Redis-based sessions store session data in Redis, which allows for better scalability and performance, especially in distributed environments.

To configure Redis-based sessions, add the following dependencies to your pom.xml:

**XML**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-redis</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.session</groupId>

<artifactId>spring-session-data-redis</artifactId>

</dependency>

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

Then, configure Redis in your application.properties:

spring.redis.host=localhost

spring.redis.port=6379

Enable Redis session management in your configuration class:

**Java**

@Configuration

@EnableRedisHttpSession

public class HttpSessionConfig {

@Bean

public LettuceConnectionFactory connectionFactory() {

return new LettuceConnectionFactory();

}

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**4. Session Cookies**

By default, Spring Boot uses cookies to store the session ID on the client side. You can customize the session cookie settings in your application.properties:

server.servlet.session.cookie.name=MYSESSIONID

server.servlet.session.cookie.max-age=3600

In Spring Boot applications, context data sharing can be achieved through various mechanisms. Here are some common approaches:

**1. ApplicationContext**

The ApplicationContext is the central interface to provide configuration for an application. It is used to manage beans and share data across different parts of the application. You can define beans in the ApplicationContext and access them wherever needed.

Example:

**Java**

@Configuration

public class AppConfig {

@Bean

public MyBean myBean() {

return new MyBean();

}

}

@Service

public class MyService {

@Autowired

private MyBean myBean;

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**2. @Value Annotation**

You can use the @Value annotation to inject values from the application properties into your beans. This allows you to share configuration data across different components.

Example:

**Java**

@Value("${my.property}")

private String myProperty;

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**3. Environment**

The Environment interface provides access to environment properties, allowing you to share configuration data across different parts of the application.

Example:

**Java**

@Autowired

private Environment env;

public void someMethod() {

String myProperty = env.getProperty("my.property");

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**4. Session Attributes**

For web applications, you can use session attributes to share data between different HTTP requests. Spring provides the @SessionAttributes annotation to manage session attributes.

Example:

**Java**

@Controller

@SessionAttributes("myAttribute")

public class MyController {

@ModelAttribute("myAttribute")

public MyAttribute myAttribute() {

return new MyAttribute();

}

@RequestMapping("/somePath")

public String someMethod(@ModelAttribute("myAttribute") MyAttribute myAttribute) {

// Use myAttribute

return "viewName";

}

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**5. Request Attributes**

For sharing data within a single HTTP request, you can use request attributes. Spring provides the RequestContextHolder to access the current request attributes.

Example:

**Java**

@RequestMapping("/somePath")

public String someMethod(HttpServletRequest request) {

request.setAttribute("myAttribute", new MyAttribute());

MyAttribute myAttribute = (MyAttribute) request.getAttribute("myAttribute");

return "viewName";

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**6. Custom Context Holder**

You can create a custom context holder to share data across different parts of the application. This is useful for sharing data that is not tied to a specific request or session.

Example:

**Java**

public class CustomContextHolder {

private static final ThreadLocal<MyContext> contextHolder = new ThreadLocal<>();

public static void setContext(MyContext context) {

contextHolder.set(context);

}

public static MyContext getContext() {

return contextHolder.get();

}

public static void clearContext() {

contextHolder.remove();

}

}